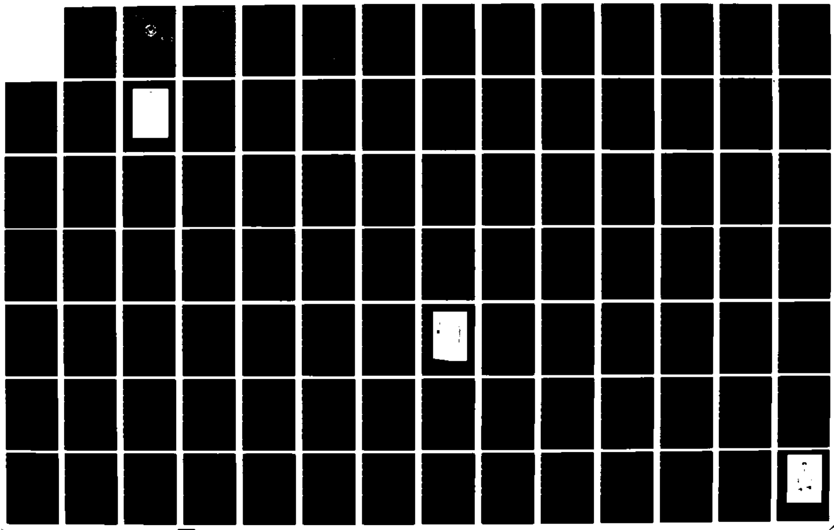
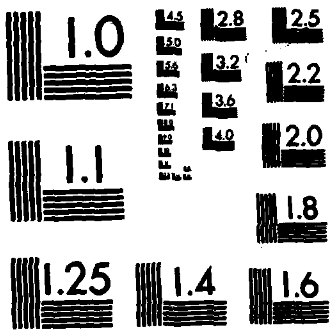
AD-A125 239 A MICROPROCESSOR CONTROLLED AUTONOMOUS SENTRY ROBOT(U) 1/2
NAVAL POSTGRADUATE SCHOOL MONTEREY CA H R EVERETT
OCT 82

UNCLASSIFIED F/G 17/5 NL




1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

2

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A MICROPROCESSOR CONTROLLED
AUTONOMOUS SENTRY ROBOT

by

Hobart R. Everett

October 1982

Thesis Advisor: R. E. Newton

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. A125239	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Microprocessor Controlled Autonomous Sentry Robot		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; October 1982
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Hobart R. Everett		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE October 1982
		13. NUMBER OF PAGES 161
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Aprpoved for public release; distribution unlimite'.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Sentry
Robot

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A microprocessor controlled prototype robot was designed and built to perform as an autonomous sentry and serve as a test vehicle for evaluation of appropriate sensors and their associated interface circuits.

A ten channel near-infrared proximity detection system was developed for use in collision avoidance, with moderate range

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

navigational planning incorporated through use of a sonar system operating in conjunction with a long range near-infrared detector.

The system was provided with a means of locating and connecting with a free standing recharging station when internal sensors detected a low battery condition.

A software structure was created to provide supervisory control of the prototype and produce a reasonably intelligent process of goal achievement through execution of ordered sequences, with provision to deal with unexpected events of higher priority.

A Microprocessor Controlled Autonomous Sentry Robot

by

Hobart R. Everett
Lieutenant Commander, United States Navy
B.E.E., Georgia Institute of Technology, 1973

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
October 1982

Author Hobart Ray Everett Jr

Approved by: R. E. Newton
Thesis Advisor

[illegible]
Second Reader

[illegible]
Chairman, Department of Mechanical Engineering

William M. Tolles
Dean of Science and Engineering

ABSTRACT

A microprocessor controlled prototype robot was designed and built to perform as an autonomous sentry and serve as a test vehicle for evaluation of appropriate sensors and their associated interface circuits.

A ten channel near-infrared proximity detection system was developed for use in collision avoidance, with moderate range navigational planning incorporated through use of a sonar system operating in conjunction with a long range near-infrared detector.

The system was provided with a means of locating and connecting with a free standing recharging station when internal sensors detected a low battery condition.

A software structure was created to provide supervisory control of the prototype and produce a reasonably intelligent process of goal achievement through execution of ordered sequences, with provision to deal with unexpected events of higher priority.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

The application of a microprocessor as a dedicated controller for a complex mechanical system eventually leads those concerned with the initial design or operation of the end result into the rapidly growing and not always understood field of robotics. This venture may be very brief, or considerably involved, depending both on the desired system capabilities, and the willingness of the designer to commit more and more functions to computer control. As required human intervention decreases, the machine's abilities must simultaneously increase, giving rise to what is commonly referred to as 'artificial' intelligence. There exists at the one end simply a pre-programmed dedicated controller able to repeatedly execute the most complex of instructions and effect the motion of actuators, valve positions, motor speeds, etc. On the other end of the spectrum, however, there are evolving machines that can function on their own, evaluating their changing environment, and reacting as needed to carry out their intended tasks with no human intervention in such a way that they truly appear 'intelligent'. The obvious question arising is "At what point do these machines become robots?". Unfortunately, there is no concise, universally agreed upon definition of a robot, or a means of identifying the transitio point No attempt will be made to add another definition to the many already in existence, as no purpose would be served in so doing, and it is doubtful that there will exist any confusion associated with the use of the term robot in the following pages.

The research presented in this thesis required the construction of a microprocessor controlled mechanical system to serve as a development and demonstration platform, and an actual robot was desired to fill this role because of its obvious impact already felt in many areas. Therefore such a system was designed and constructed to be used strictly as a learning tool, with no attempt to provide the prototype with an outer protective skin. All circuits and interfaces as well as the mechanical parts were to remain as open and accessible as possible without degradation of performance or risk of physical damage. For purposes of illustration this robot was given the assigned task of serving as a home sentry, patrolling in a normal household environment without human intervention. It was to be equipped to detect potential dangers such as smoke, fire, toxic gas, flooding conditions, or intrusion, and then effect the necessary response. The system was not intended to be completely autonomous at first, but to evolve to that stage over a period of time as new techniques were developed, additional sensors added, and the concepts of artificial intelligence explored by the designer.

Once the decision is made as to the overall task (or combination of several smaller tasks) a robot is to perform, the actuators must then be provided to enable it to mechanically accomplish that task. The next step involves selection of the microprocessor(s) which will provide the control, and the sensors required to furnish the microprocessor with its needed information. As the design considerations begin to materialize the question of how much human interaction is needed or desired must be readdressed, and then attention turns to which points internal to the system must be monitored, such as voltage levels, temperatures, current flow, etc. The design can then be successively improved through an iterative process until a reasonable solution is reached for implementation on the prototype.

If the robot is to carry its own power source then energy conservation becomes critical in order to achieve a practical duty cycle relative to the time required to recharge the battery pack. Circuits which are powered down when not needed often will affect other systems which are still energized and any interconnection must take this into account. Backup or supplementary systems must be structured so as not to provide conflicting information to the microprocessor, and any change-over must be smooth and orderly. Individual tasks must be prioritized to allow those of greater urgency to interrupt the less important routines which may be in progress, in such a way that the CPU can pick up where it left off or cancel the remainder upon return, as appropriate. Collision avoidance routines must be set up so as to maximize the use of the available sensory information, and provision must be made to react to impending collisions that arise as a result of evasive action initiated by a previous threat of collision. The concept of interrupts must be thoroughly understood by the designer, with much attention given to the details as applicable to the microprocessor chosen if the system's capabilities are to be utilized in the most effective manner.

A. DESIGN CONSIDERATIONS

As indicated, the primary purpose of this robot is to serve as a mobile platform for research and experimentation in the areas of artificial intelligence, computer interface techniques, speech synthesis and recognition, and mechanical and electrical design. The initial design provided for the following:

1. Chassis and body framework with a tricycle wheelbase featuring a driven steerable front wheel.
2. A rotating head assembly mounted on the body trunk, positionable up to 100 degrees either side of centerline.
3. Speech synthesis for communications with operator.
4. Optical photocell array located in head for locating and tracking homing beacon on recharging station.
5. Single transducer SONAR system for determining range to obstacles in immediate path.

6. Multi-element near-infrared collision avoidance system for object detection in first and fourth quadrants relative to centerline.

7. Contact bumpers and feelers for collision detection.

8. Numerous sensors for intrusion detection and home surveillance.

9. Complete software development system with hardcopy printer.

10. Provision to allow operator to request control from CPU for performance of trouble shooting routines or to request a specific behavior pattern subroutine.

The size of the prototype was arrived at through consideration of several design requirements. The overall height was chosen so as to allow the machine to 'see' over most obstructions likely to be encountered in a home environment, to facilitate location of the recharging station, yet not so high as to preclude the fabrication of two uprights from a standard six foot length of aluminum angle stock. A body trunk height of 36 inches, with an additional 2 inches for floor clearance, effectively situates the optical photocell array at 44 inches above the floor (see Figure 1). The additional sensors mounted on top of the head extend the height to 57 inches, and the receiving whip antenna reaches up to a total system height of 62 inches.

The width of the base was set at 15 inches to allow sufficient room for the drive motors attached to either side of

Figure 1. Photo of the Prototype Robot ROBART

the front wheel, and to provide a stable foundation for the body trunk. The two rear wheels are recessed into the side of the rectangular base to provide a smooth side panel relatively free of protrusions which would enhance the possibility of impact with surrounding objects. The base length is 26 inches from front to rear bumper, and the relatively narrow profile allows for easy navigation between obstacles and through doors. The tricycle wheelbase provides a minimum turn radius of twelve inches measured to the drive wheel of the vehicle, which translates to a required free radius of 21 inches in which to perform a turn without impact. The maximum turn angle is 80 degrees either direction.

The tricycle wheel base was chosen for use on this robot primarily for comparison with a system employed on a previous prototype, which consisted of two separately controlled drive motors attached to the rear wheels, and caster type idlers used in front. The reader should not assume that the steerable driven front wheel is preferred over alternative methods (see Section V).

B. MICROPROCESSOR SELECTION

Two microprocessors are used on the prototype robot, one to provide supervisory control and another totally dedicated to speech synthesis. The first of these, a 6502 microprocessor, is located on a SYM-1 single board computer manufactured by Synertek Incorporated of Santa Clara, CA. A functional block diagram of the SYM-1 computer is shown in Figure 2. The SYM was chosen as the primary controller for the prototype due to the extensive hardware made available on a single board, the availability of a superb Assembler/ Editor for software development, and a relatively low total package price.

The 6502 utilizes a 16-bit address bus and an 8-bit data bus, and provision is made to utilize 4 Kilobytes of Random Access Memory (RAM) on board, as well as 28 Kilobytes of Read Only Memory (ROM). Three 6522 Versatile Interface Adapters (VIA) and one 6532 Peripheral Interface Adapter (PIA) are available on-board and together provide a total of 71 Input/Output (I/O) lines, making the SYM ideal for robotic applications. Throughout the rest of the text, the three Versatile Interface Adapters will be referred to as 6522-1, 6522-2, and 6522-3. Their respective port assignments are given in Appendix A. Off-board expansion is provided for through a 44-pin Expansion Connector, and 32 Kilobytes of additional RAM are incorporated through the addition of Beta Computer Devices 32K RAM Expansion Board.

The SYM-1 computer is mounted at mid-height on the front of the prototype, forming the heart of the electronics. It functions primarily as a dedicated controller, but can be borrowed for interface to a Synertek KTM-2 terminal through an RS-232 connection, thus greatly expanding the practicality of the overall setup. The robot remains motionless beside

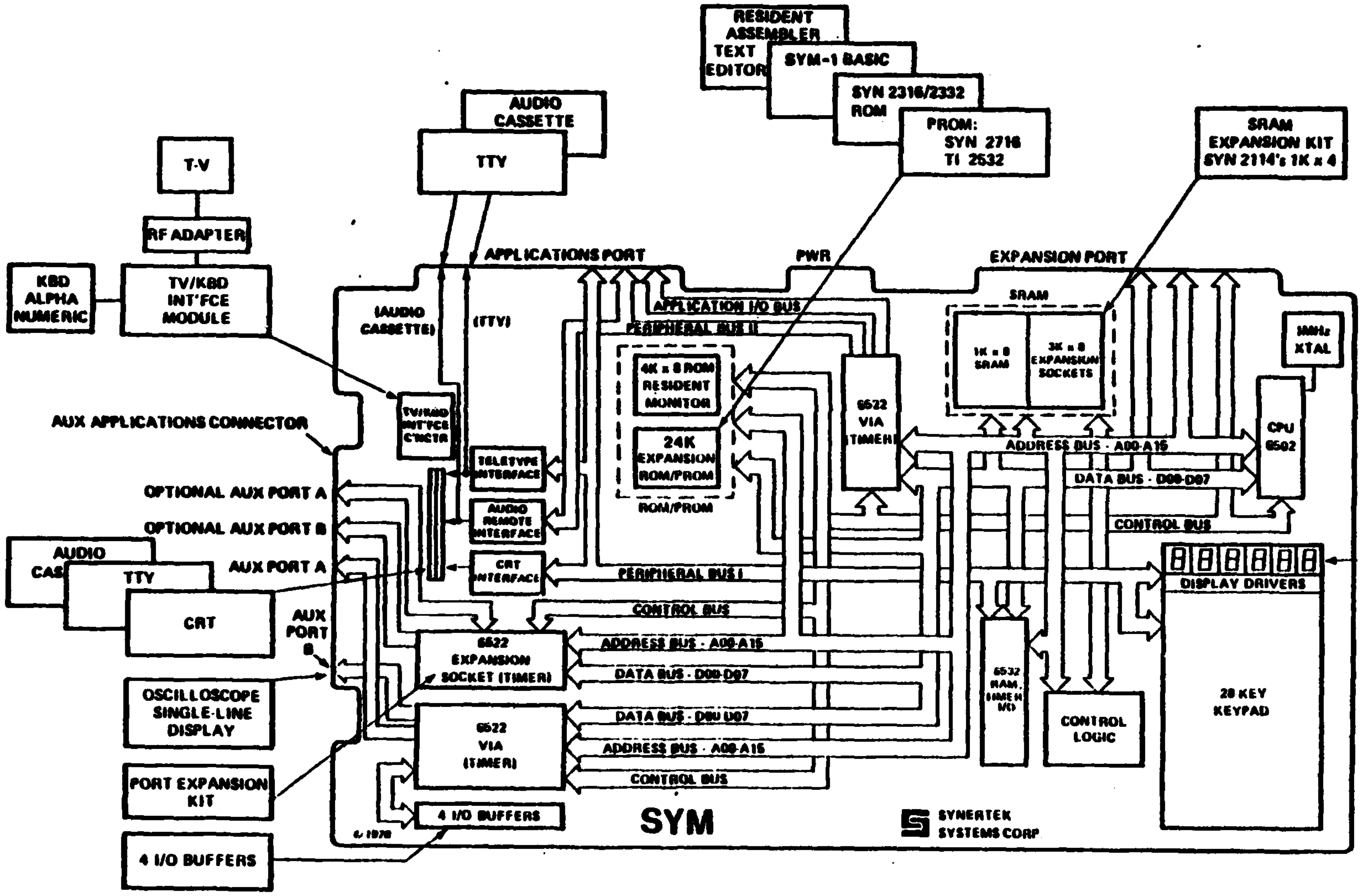


Figure 2. Functional Block Diagram of SYM-1 (courtesy of Synertek Systems Corp.)

the terminal stand while the SYM is under KTM control, and in exchange receives supplemental power over the same DB-25 connector that provides the RS-232 link. Once released by the operator, the CPU first verifies that this circuit has been disconnected, and then the robot proceeds under its own control on battery power. In the event the connecting cable has not been removed, operator assistance will be requested through speech synthesis.

Speech synthesis is implemented through National Semiconductor's Digitalker DT1050 synthesizer chip, with two sets of vocabulary instructions stored on EPROMs for a total vocabulary of 280 words. (A third set will soon be available.) A fixed vocabulary was preferred over an unlimited vocabulary created through repeated use of phonemes due to the greatly decreased demand on the host microprocessor in terms of both execution time and memory space. This also reduces the complexity of voice outputting a response to changing conditions as encountered by the robot when operating under its own control. These considerations are especially important when designing a system that is based around a single microprocessor. The SYM-1 CPU is interfaced to the Digitalker DT1050 through two parallel I/O ports, one of which supplies the EPROM starting address for the instructions needed to generate the desired word. The EPROMs are addressed by the DT1050 and do not take up address space of the SYM-1. A portion of the second I/O port is used to initiate the speech output, and to detect completion of each word. The SYM-1 essentially just instructs the synthesizer to speak a particular word, and then checks later to see if the speech is complete before requesting the next word.

The Subroutines Vox1 and Vox2 request words from vocabulary lists 1 and 2, respectively, just as Vox3 will deal with those words on list 3 when its associated EPROMs become available. The hex address of the desired word is first loaded into the X Register of the SYM-1, and then the appropriate subroutine called (Vox1 or Vox2). After the desired word is identified, Subroutine Talk is called which manipulates the control lines to the DT1050 to initiate the speech, and then waits for the busy signal on PB7 of 6522-1 to clear. Subroutines Vox1d and Vox2d can be called if a slight delay is desired before outputting the word specified, for spacing between words in a sentence.

C. INTERFACE LAYOUT

When used as a dedicated controller, the SYM-1 microcomputer communicates with the outside world through its three 6522 Versatile Interface Adapters and the 6532 Peripheral Interface Adapter. These each contain two parallel eight-bit input/output ports (Port A and Port B), which can be used to read sensory information or send commands to external circuitry. Each 6522 contains a total of sixteen internal eight-bit data registers, and is structured as shown in Figure 3. The organization of the 6532 is very similar. For the purposes of this discussion the differences are insignificant and will not be addressed.

As configured on the SYM, all sixteen registers appear to the CPU as specific memory locations within the 64 kilobyte address space. The CPU reads or writes data from these registers on the VIAs as it would from any memory location. The assigned address locations for the specific registers in each of the four input/output devices are given in the SYM-1 Reference Manual [Ref. 1].

For both Port A and Port B there is associated an eight-bit register referred to as the Input/Output Register, and it is this register that the CPU actually reads from or writes to when communicating with the outside world. Each of its eight bits is directly associated with a hardware connection to one of the expansion plugs on the SYM-1 board. In the output mode, data can be loaded into this register, and these associated lines will assume the TTL logic level dictated by the subsequent register contents. For each bit that is high, the output line will go high, whereas each low bit will cause its associated line to go low, as shown in Figure 4. These registers can be incremented, decremented, or rotated by assembly language instructions, with the voltage levels on the output lines reacting accordingly.

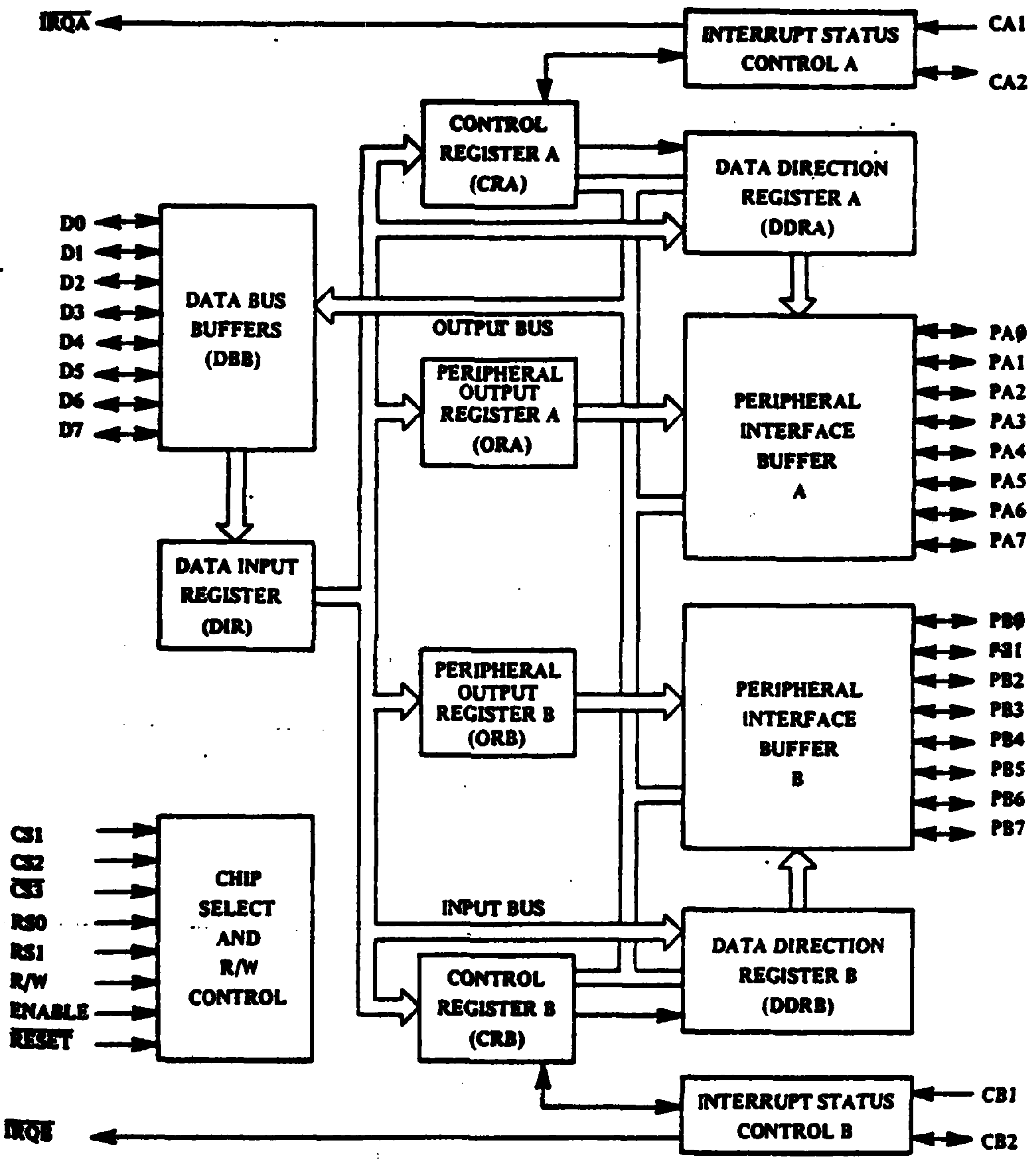

IRQA
INTERRUPT STATUS CONTROL A
CA1
CA2
CONTROL REGISTER A (CRA)
DATA DIRECTION REGISTER A (DDRA)
D0
D1
D2
D3
D4
D5
D6
D7
DATA BUS BUFFERS (DBB)
OUTPUT BUS
PERIPHERAL OUTPUT REGISTER A (ORA)
PERIPHERAL INTERFACE BUFFER A
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
DATA INPUT REGISTER (DIR)
PERIPHERAL OUTPUT REGISTER B (ORB)
PERIPHERAL INTERFACE BUFFER B
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
CS1
CS2
CS3
RS0
RS1
R/W
ENABLE
RESET
CHIP SELECT AND R/W CONTROL
INPUT BUS
DATA DIRECTION REGISTER B (DDRB)
CONTROL REGISTER B (CRB)
INTERRUPT STATUS CONTROL B
CB1
CB2
IRQB

Figure 3. 6522 VIA Internal Architecture (courtesy of Synertek Systems Corp.)

These output lines can be directly interfaced to control solenoids, motors, lamps, etc., according to preprogrammed instructions executed by the 6502.

In the input model the TTL voltage levels present on the lines associated with the Input/Output Register dictate the individual bit values within the register. Thus the state of a line can be determined by reading the register, and masking off all but the desired bit with logical operations performed within the accumulator by appropriate 6502 commands.

The two Data Direction Registers associated with each 6522 VIA determine the mode of operation for each individual bit in the Input/Output Registers. If the processor writes a 0 into a particular bit of the Data Direction Register, that same bit will be configured as an input on the Input/Output register of the same port (see Figure 4). Conversely, writing a 1 will cause the pin associated with that bit to act as an output. Any combination of inputs and outputs is possible on the same port.

The remaining registers used on the 6522 VIAs are concerned with shift register operation, event timers, and interrupt processing, and an adequate discussion is not possible here. The reader is referred to Scanlon, "6502 Software Design" [Ref. 2], and Zaks, "6502 Applications Book" [Ref. 3] for excellent coverage of these details.

As the initial design of the prototype begins to materialize, it soon becomes apparent that the 71 I/O lines

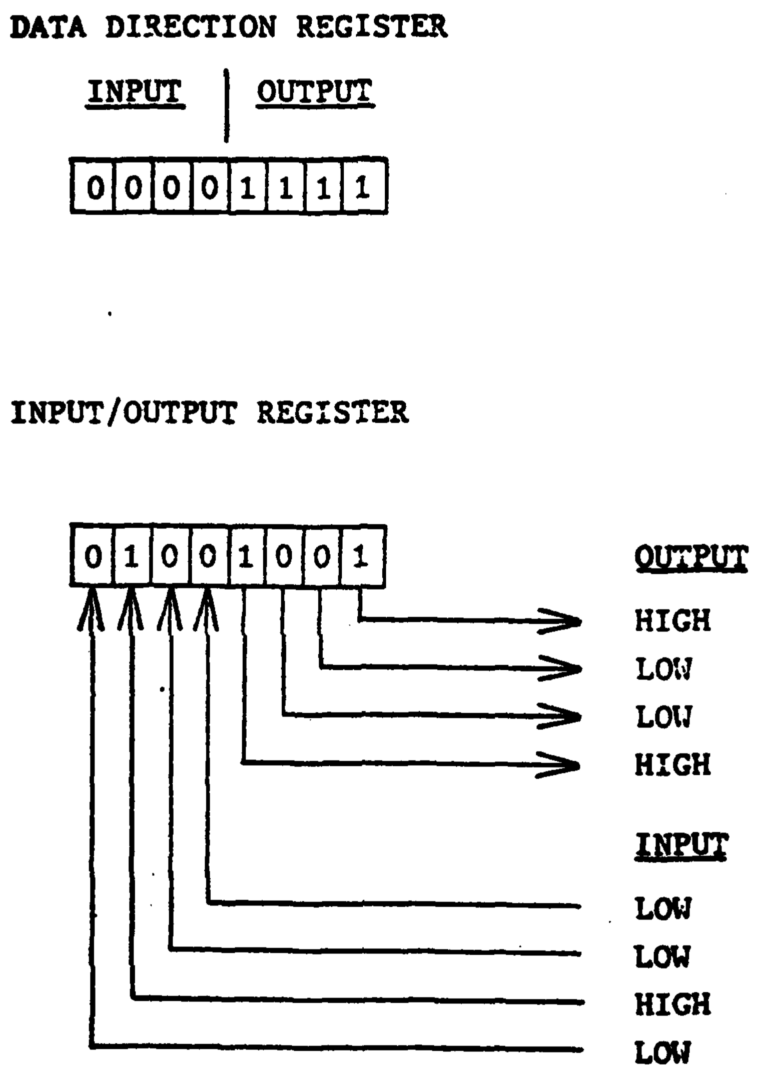


Figure 4. Data Direction Register and Input/Output Register. Bits set high in the Data Direction Register configure corresponding bits in Input/Output Register as outputs.

available on the SYM-1 microcomputer are insufficient, due to the vast number of inputs and outputs needed for autonomous control of a mobile system. Thirteen I/O lines are needed for communication with the speech synthesis microprocessor alone, eight for head and drive wheel position control, five for an operator control interface, and sixteen for tactile sensors, to name but a few.

Therefore a four line address bus was created to serve the six interface boards which connect the CPU to the various sensors and outputs under its control. This address bus is driven by PB0 - PB3 of 6522-1 (see Appendix A), and in turn drives three 74150 sixteen-input Data Selectors, two 74154 sixteen-output Data Distributors, and one 7475 four-bit latch (see Figure 5). This yields an additional 84 I/O lines, but at the expense of 10 of the original 71 I/O lines. If needed, additional groups of 16 lines each could be added at a cost of 1 original I/O line per group.

The three Data Selectors are simultaneously driven by the four line address as the CPU sets PB0 - PB3, with the complement of the selected input appearing on the selector output, pin 10. A detailed description of 74150 and 74154 operation is given by Lancaster in his "TTL Cookbook" [Ref. 4]. For ease in programming, the selector outputs are again inverted before being read by appropriate inputs on the 6522 VIAs, so that the VIA sees the same logic level (high or low) as seen by the selector input. This inversion is done by a spare

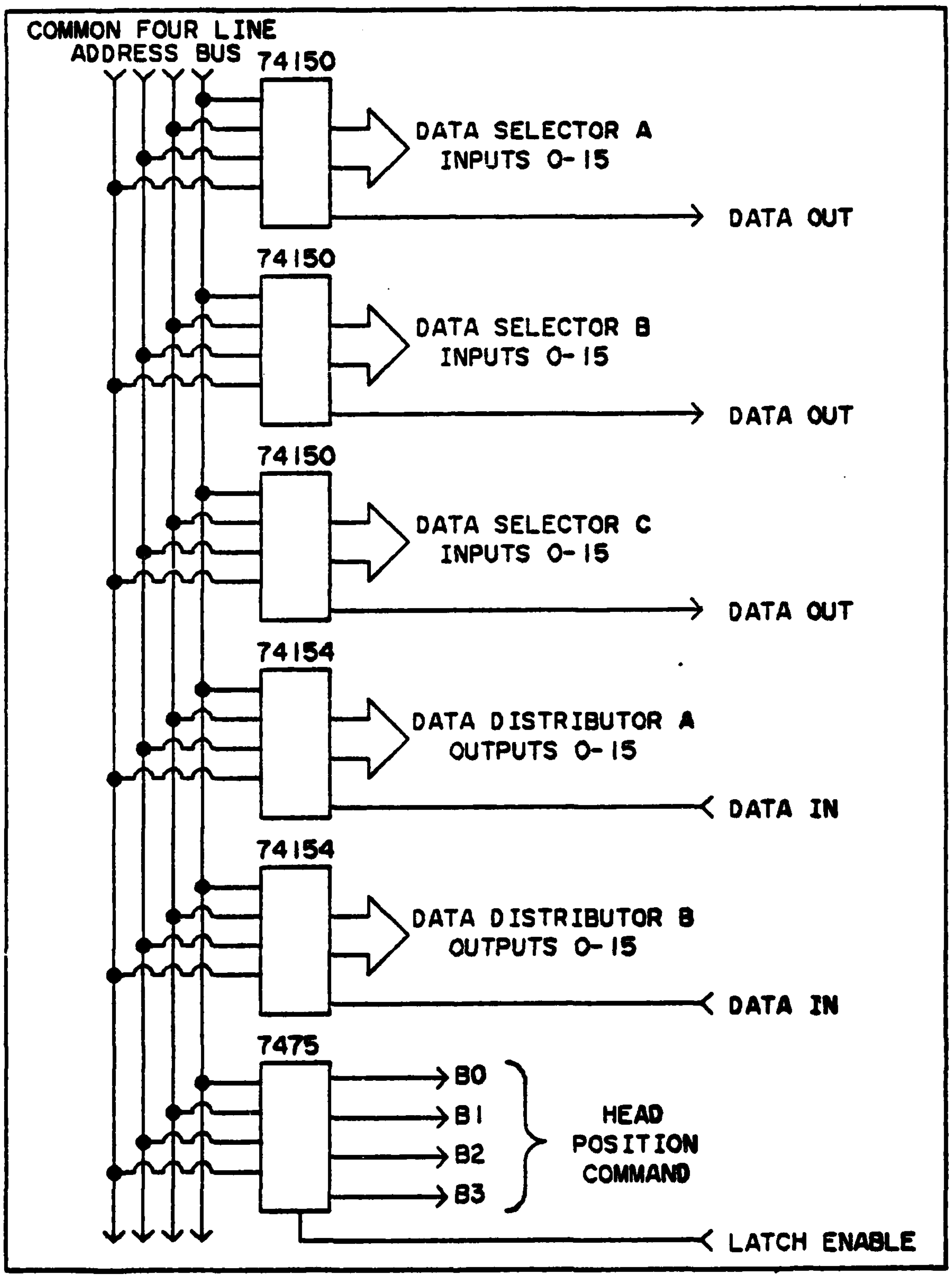

COMMON FOUR LINE ADDRESS BUS
74150
DATA SELECTOR A INPUTS 0-15
DATA OUT
74150
DATA SELECTOR B INPUTS 0-15
DATA OUT
74150
DATA SELECTOR C INPUTS 0-15
DATA OUT
74154
DATA DISTRIBUTOR A OUTPUTS 0-15
DATA IN
74154
DATA DISTRIBUTOR B OUTPUTS 0-15
DATA IN
7475
B0
B1
B2
B3
HEAD POSITION COMMAND
LATCH ENABLE

Figure 5. Interface Layout and Four-Line Address Bus

comparator on the quad LM339, used to implement a sixteen input NOR gate. A schematic of Data Selector A is shown in Figure 6.

The CPU therefore must set the value of the desired input number onto the four line address, and then read the appropriate selector output over the corresponding 6522 VIA input (see Appendix A). As an example, to read the Target Input used in homing on the recharging station, the CPU must set the address lines to binary 0101 to select input number 5 on all three Data Selectors, and then read the output of Data Selector C over PA6 on 6522-3. Subroutines ReadA, ReadB, and ReadC take care of manipulating the address and data lines for ease in programming, with the desired input number loaded into the X Register before the appropriate subroutine is called.

Data Selector A is entirely dedicated to tactile and proximity sensors, and is interrupt capable (see Section II-B). Data Selector B also is interrupt capable, and handles all alarms and internal circuitry check points. Data Selector C is used to read miscellaneous inputs and has no interrupt capability. A listing of all selector inputs is given in Appendix B.

The two sixteen-output 74154 data distributors are also driven by the four line address bus, but require three additional control lines, referred to as Data 1, Data 2, and Data 3 (see Appendix A). These three lines are normally

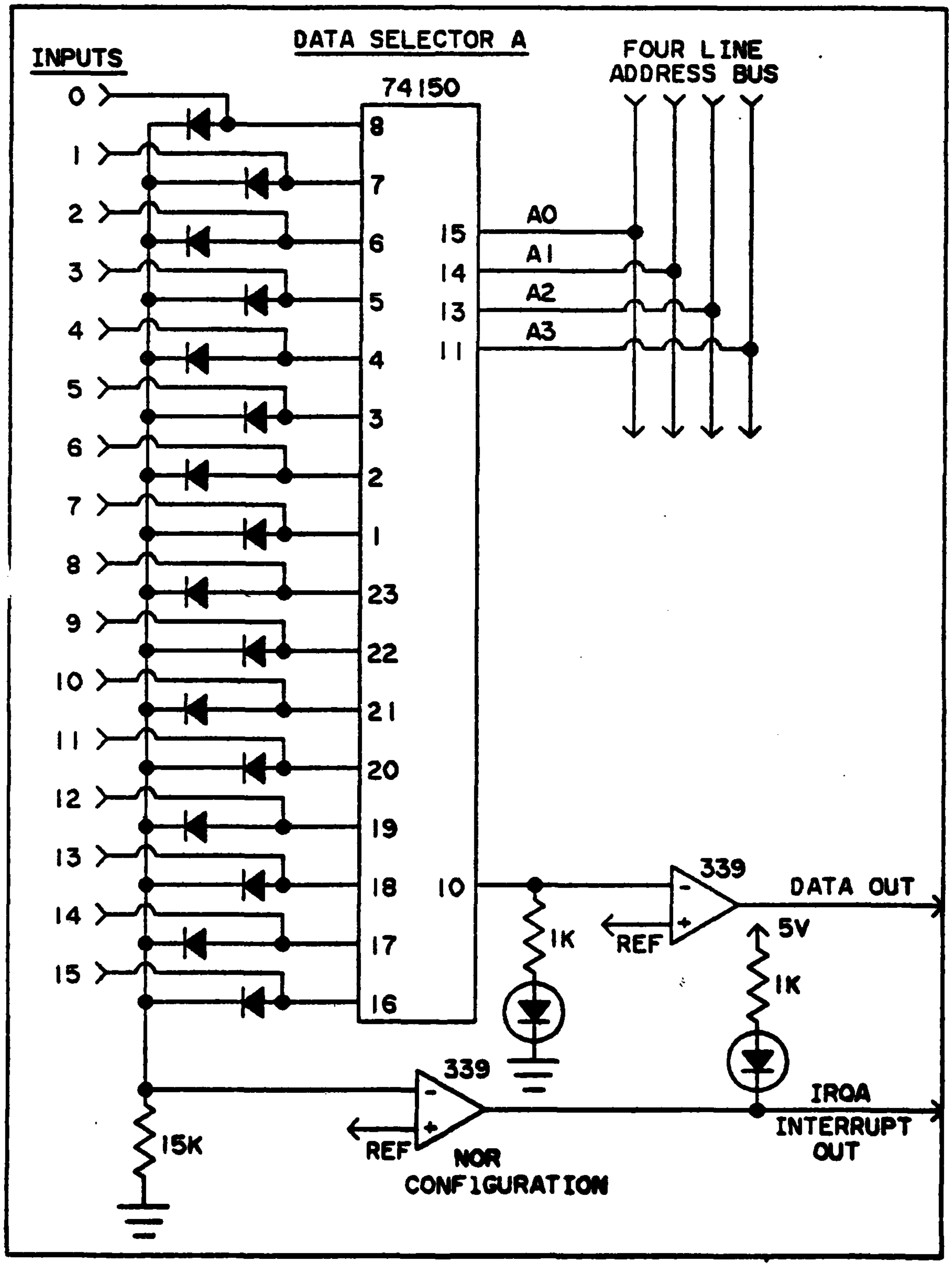


Figure 6. Data Selector A. Spare comparator on the quad LM339 is used as an inverter to invert the logic at the selector output, pin 10. Data Selectors B and C are similarly configured except that C has no interrupt capability.

maintained in the high state. Data 1 and Data 2 provide the inputs to Data Distributor A and Data Distributor B, respectively. These lines can be sent low to pull down the desired distributor output. All sixteen outputs are normally high, except the selected output, which follows the applied input. Thus if the inputs are kept high, no outputs will change state regardless of the status of the four line address. This is important as this address also services other devices.

If the CPU desires to momentarily pull low an output on Distributor B, it first sets the appropriate binary value on the four line address bus (0011 for output number 3, for example), and then momentarily sends Data 2 low. Output 3 on Distributor B will also go low, but output 3 on Distributor A, although selected, will be unaffected, since it follows Data 1. It is important to note, however, that this system can only be used to strobe outputs on Selector B, and that they cannot be held low for any length of time without tying up the CPU. Therefore these outputs are used only as negative-going triggers to initiate timing sequences or actions subsequently controlled by other circuitry.

The third control line is used with Distributor A to overcome this problem and provide a means for latching the output high or low. As shown in Figure 7, each output on Distributor A is used to clock a positive-edge-clocked D-type flip-flop. These flip-flops all have their D inputs commonly connected to Data 3. The output of any flip-flop will

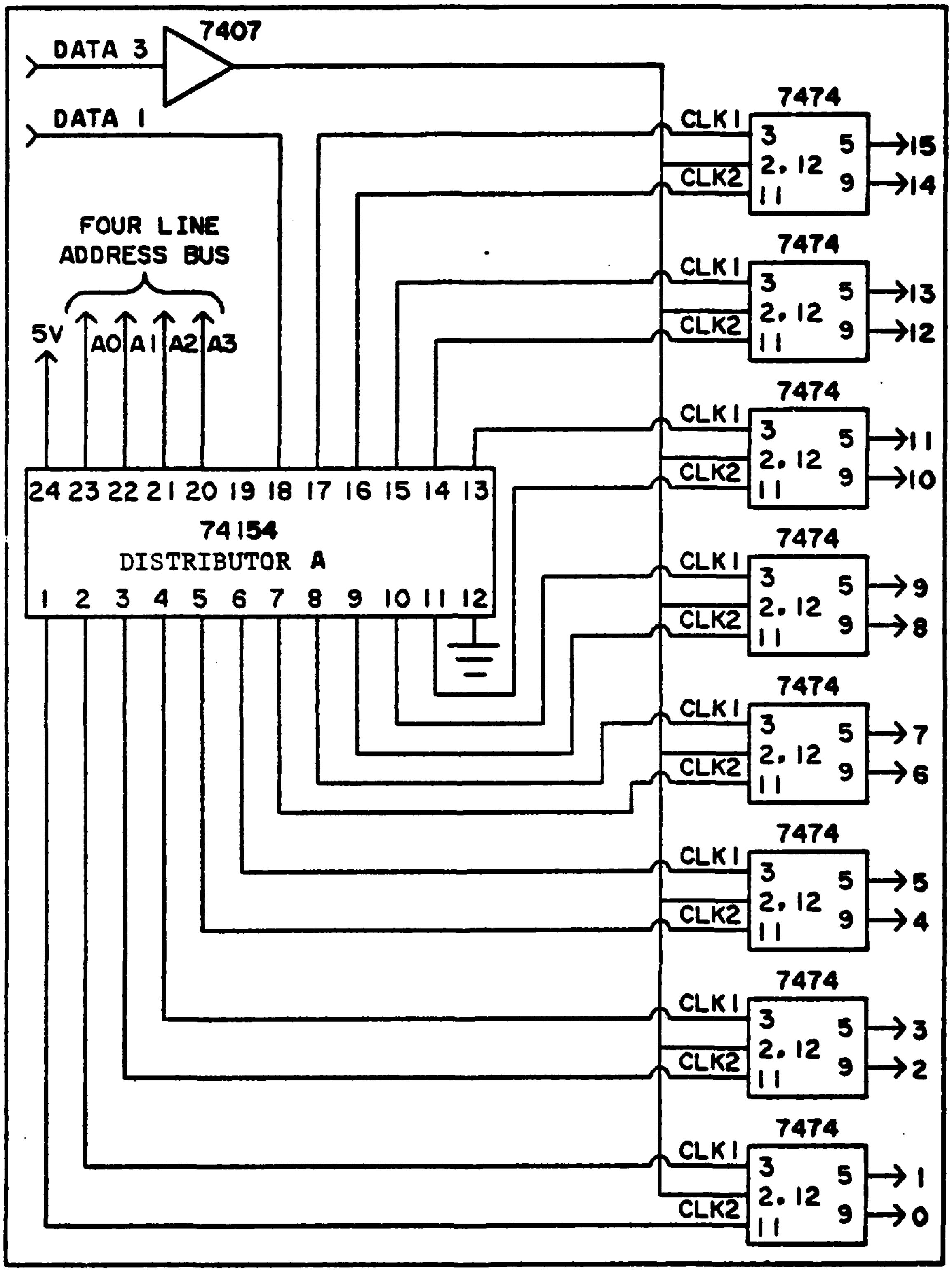


Figure 7. Data Distributor A. Distributor outputs are used to clock 7474 flip-flops for latching the logic state present on Data 3.

therefore assume the logic level of Data 3 if and only if that flip-flop is clocked by Distributor A. The clocking sequence involves setting the appropriate address to select the desired flip-flop, with Data Lines 1-3 high. Data 3 is then set to the desired output logic level (high or low). Data 1 is then strobed to clock the flip-flop, and its output assumes and holds the desired state dictated by Data 3. No other flip-flops will change state, as only the selected flip-flop was clocked. For a complete description of the 7474 flip-flop operation, the reader is again referred to the "TTL Cookbook" [Ref. 4].

This rather complicated manipulation of address and control lines is all done by Subroutines Pin.hi and Pin.lo, which respectively set the address according to the contents of the X Register and then send the appropriate output high or low. For example, to turn on the robot's spotlights, the programmer would merely load the X Register with hex 01, and then call Subroutine Pin.hi, which would set flip-flop number 1 output to high, enabling the spotlights. A listing of all Distributor Outputs is given in Appendix B.

The final device serviced by the four line address bus is a 7475 quad latch, used to store a four bit command for head positioning circuitry to be discussed later. This latch is level sensitive, and controlled by PB7 of 6522-2, referred to as Latch Enable. When this line is high, the latch contents will reflect the value of the four line address bus and subsequently hold that value when Latch Enable goes low. Thus any four-bit number can be latched into the register as a head position command.

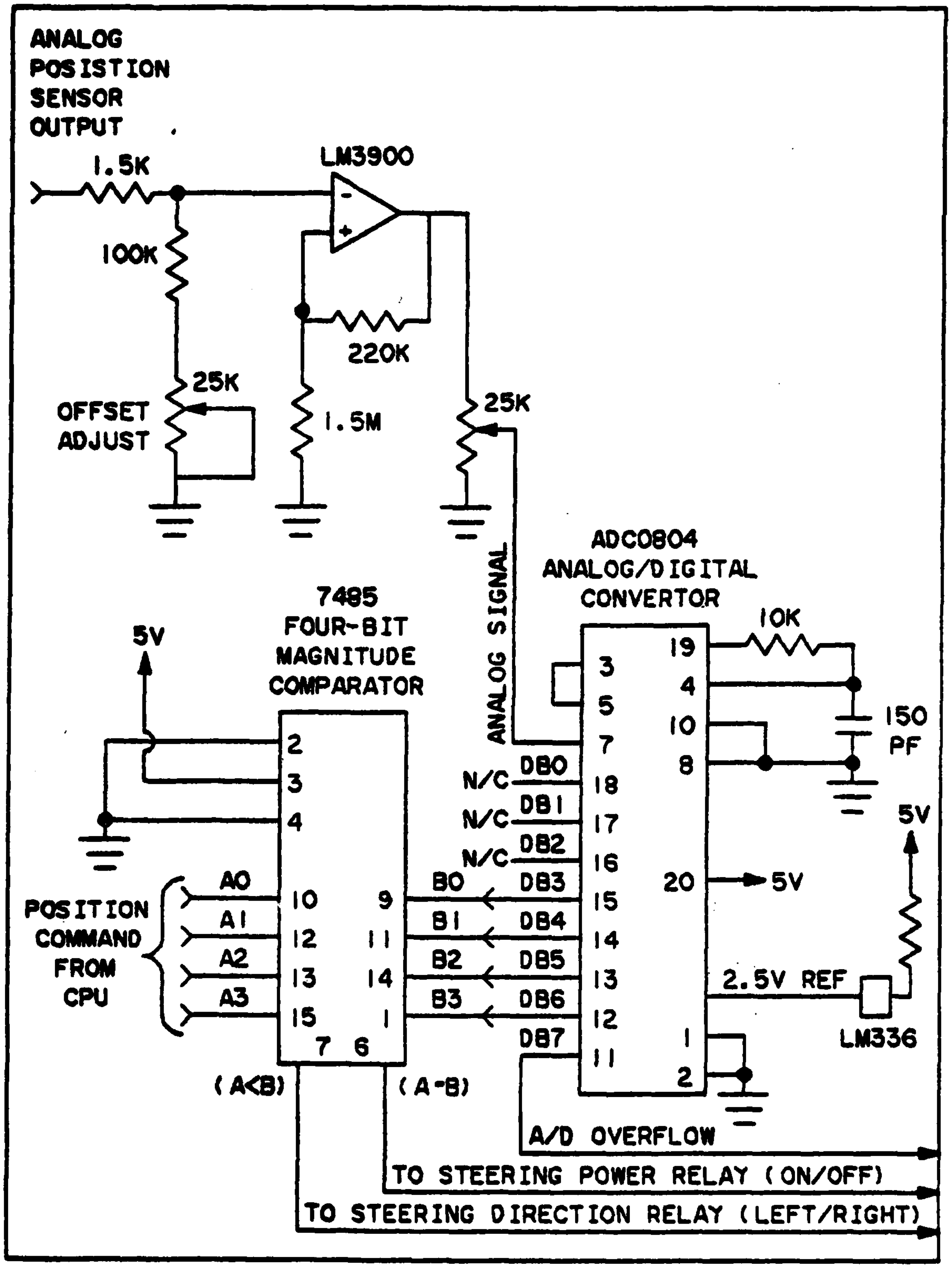
D. DRIVE AND STEERING CONTROL

The prototype robot is propelled by two surplus gear motors originally designed for use as valve actuators. Each has a separate forward and reverse winding, with a permanent magnet field. Output shaft speed at 12 volts DC is 10 RPM under full load, yielding a forward velocity of approximately 16 feet per minute. The two drive motors are mounted on either side of a single front wheel, attached to what will be referred to as the drive wheel support cage. This cage in turn is supported by a vertical steering column and thrust bearing in such a way as to be positionable by a steering motor up to 80 degrees either side of centerline. The entire drive assembly thus pivots around the steering column. Position is sensed by a belt driven potentiometer mounted directly to the rear of the column. The steering motor is identical to the two motors used for propulsion.

The sensing potentiometer used for position feedback is wired as a voltage divider across a carefully regulated 5V supply, and positioned so as to provided a linear output ranging from 0V to 3.7V, as the wheel turns from right to left. This voltage is fed to an operational amplifier for isolation purposes, and the amplifier output applied across a 25 K potentiometer used to set the maximum output to 2.5 volts (wheel full left). The output from this second potentiometer is fed directly to a National Semiconductor ADC0804 analog to digital convertor (see Figure 8) located on Interface Board Number 2. This A/D converter is supplied in a 20 pin package, and in this application operates in the free running mode, with eight bits of resolution in a parallel output. The chip is designed for an input voltage swing of 0V to 5V, and so the most significant bit is not used. This in effect creates an A/D converter with a range of 0V to 2.5V, but with only seven bits of resolution. The unused bit should remain low during normal operation, and is wired to generate an interrupt if it ever goes high, indicating the system is no longer correctly calibrated, resulting in an A/D overflow.

Due to monetary constraints, a rather simple positioning control system was chosen for use on the prototype, with substitution of a more sophisticated version easily achievable if later desired. This simple control system required only four bits of resolution, effectively creating 16 discrete position points throughout the range of motor travel, approximately every 10 degrees. Therefore, the three least significant bits of the ADC0804 are not used.

The four bits from the A/D converter are fed to the B inputs of a 7485 four-bit magnitude comparator, as shown in Figure 8. The desired steering position is fed to the A inputs directly from 6522-2, PA0 - PA3. The comparator



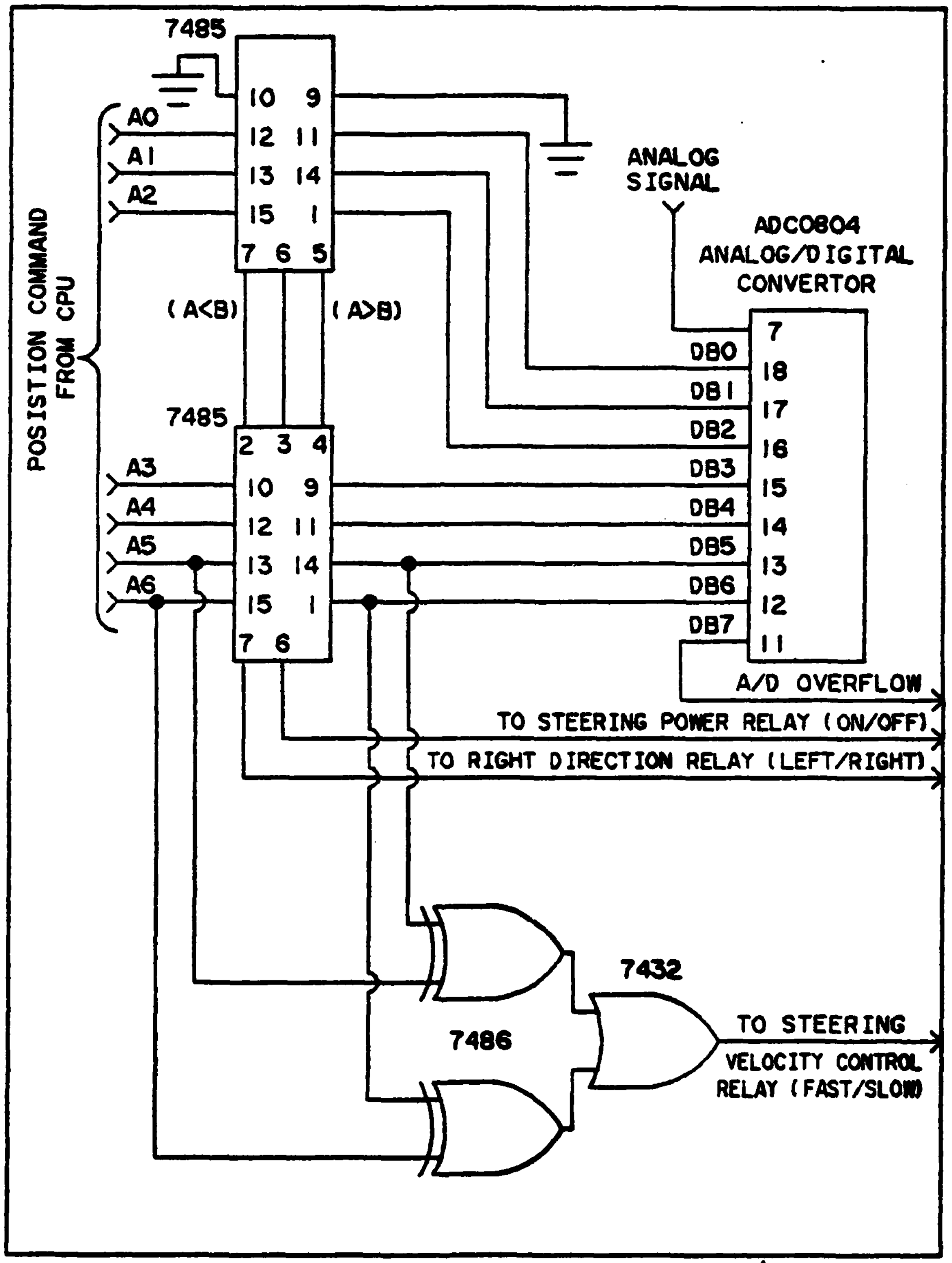
ANALOG POSISTION SENSOR OUTPUT
1.5K
LM3900
100K
220K
25K
OFFSET ADJUST
1.5M
25K
ANALOG SIGNAL
ADC0804 ANALOG/DIGITAL CONVERTOR
10K
150 PF
7485 FOUR-BIT MAGNITUDE COMPARATOR
5V
5V
5V
N/C DB0
N/C DB1
N/C DB2
DB3
DB4
DB5
DB6
DB7
B0
B1
B2
B3
A0
A1
A2
A3
POSITION COMMAND FROM CPU
2.5V REF
LM336
(A<B)
(A=B)
A/D OVERFLOW
TO STEERING POWER RELAY (ON/OFF)
TO STEERING DIRECTION RELAY (LEFT/RIGHT)

Figure 8. Steering Position Control Schematic

compares the two numbers and indicates via its digital outputs if they are equal, or else which is the larger. The steering motor can be directly controlled by these same outputs, as was done by DaCosta [Ref. 5] in a similar arrangement which served as the basis for this design. The steering motor is energized as long as the actual position is not equal to the deisred position, and its direction is determined by which of the two is greater.

As pointed out, this is a relatively crude positioning scheme, with very coarse resolution, and no velocity control. Nevertheless it proved more than adequate for use in the homing and navigational routines presented later. Considerable improvement could be obtained by ganging together two 7484 comparators, and picking up the additional three bits available from the A/D convertor, for a total of 128 discrete positions as opposed to merely 16. The upper two most significant bits could be monitored with Exclusive-Or logic gates to control the speed of the motor as well, causing it to slow down to a reduced RPM when the upper two bits matched, stopping altogether when all bits matched. A proposed schematic is shown in Figure 9.

A considerable amount of frictional damping is inherently present in this positioning system, which decreases the chances of overshoot, especially for the case of only sixteen discrete positions. As the resolution is increased, however, overshoot becomes a significant problem, and some form of



7485
10 9
12 11
13 14
15 1
7 6 5
A0
A1
A2
POSISTION COMMAND FROM CPU
(A<B)
(A>B)
7485
2 3 4
A3
A4
A5
A6
10 9
12 11
13 14
15 1
7 6
ANALOG SIGNAL
ADC0804 ANALOG/DIGITAL CONVERTOR
7
DB0 18
DB1 17
DB2 16
DB3 15
DB4 14
DB5 13
DB6 12
DB7 11
A/D OVERFLOW
TO STEERING POWER RELAY (ON/OFF)
TO RIGHT DIRECTION RELAY (LEFT/RIGHT)
7432
7486
TO STEERING VELOCITY CONTROL RELAY (FAST/SLOW)

Figure 9. Improved Design for Steering Control

velocity control becomes essential for stability. The ideal solution would perhaps be to use a small dedicated microprocessor to compare the desired position with the actual position, and control the motor velocity accordingly with pulse width modulation; continuously decreasing the motor speed as a function of position error. Single board systems suitable for this application are available for less than $100.

The tandem drive motors used for propulsion are wired in parallel to drive the steerable front wheel, and can operate in either forward or reverse directions. Due to their slow maximum speed, no attempt was made to provide velocity control. These motors were selected for their extremely low cost and high torque, and will be replaced by faster versions in a follow on robot to be built based on experiences gained through this development. The drive motors are energized by a relay controlled by PA4 of 6522-2, and their direction is similarly controlled by PA5 of the same port.

E. INTERRUPT ROUTINES

The software structure for the prototype robot utilizing the Synertek Systems SYM-1 microcomputer as a controller can be broken down into three general areas. In addition to the main code which handles overall system control, there are two sections which deal with interrupts: the Non-Maskable Interrupts (NMI) and the maskable Interrupt Request (IRQ). Coordination among these three areas is made possible by dedicating certain register locations in page zero (memory address locations $0000 - $00ff) for information transfer, and also by the fact that input/output devices can be accessed from all three areas, not just the main code. A brief introduction to the concept of interrupts is required before presenting an explanation of the two types utilized on the prototype.

An interrupt is an event whose occurrence is hardwired to force the processor to drop what it was doing and service the cause of the interrupt. No polling of devices is required until an interrupt is detected. This leads to much more efficient operation, as the CPU need not concern itself with any interrupt sources until such time as they actually require attention. The CPU is alerted to this condition by special input lines connected to the hardware that it services - 6522 Versatile Interface Adapters in the case of the SYM-1 single board computer.

There are two interrupt input lines to the 6502 microprocessor. Each can be used to halt temporarily the program under execution and cause a branch to a different location in memory. The processor then executes the program listed at this new location, which is referred to as the interrupt service routine. This action by the processor is termed responding to an interrupt. Quite often the processor branches to a specific location that contains the starting address of the interrupt service routine, and then branches again to that starting address. This allows the service routine to be located anywhere in memory, rather than being restricted to a specific address. This concept is referred to as 'vectored' interrupts, and the specific address where the processor goes to fetch the service routine starting address is known as the interrupt vector location.

Two types of interrupts are possible with the 6502 microprocessor: the Non-Maskable Interrupt (NMI) and the Interrupt Request (IRQ). The Non-Maskable Interrupt can override the lower priority Interrupt Request, and will occur whenever the NMI line (pin 6) is pulled low on the 6502. It is edge sensitive, occurring on the high to low transition of pin 6, and cannot be internally masked by the processor. An Interrupt Request, however, occurs when the IRQ line (pin 4) goes low. Unlike the NMI, the IRQ is level sensitive, which means that the processor will be interrupted as long as pin 4 on the 6502 is held low. A second difference is that Interrupt Requests can be temporarily disabled by setting a flag within the processor, called the interrupt disable bit. This bit can be set through software, and when set, causes all subsequent Interrupt Requests to be ignored. (The Assembly Language command to set this bit is SEI, and it is cleared with the command CLI.) It is important to note that this bit is set automatically by the processor when responding to an IRQ, and cleared automatically when returning from interrupt (RTI). The programmer has the option of changing its status as he so desires either within the interrupt service routine, or external to it. In any event, the condition causing the interrupt must first be dealt with before interrupts are re-enabled, or another interrupt will immediately occur, and an endless loop will result. The programmer has two options: 1) Provide for eliminating the source of the interrupt through action initiated in the interrupt service routine, verify elimination, and then return from interrupt. 2) An alternate method would be to disable the device reading the interrupt (i.e., the hardware between the source and the 6502 CPU), then return from interrupt and attend to the source. Once the condition has been cleared, the hardware which alerts the CPU to an interrupt condition can be re-enabled for future use. Both methods are used in the prototype as presented later in the text.

When a Non-Maskable Interrupt occurs, the processor will complete the instruction currently being executed before recognizing the interrupt, and then store the contents of the Program Counter and the Processor Status Register on the stack. The processor then goes to a specific address in memory ($A67A) and fetches the starting address of the Non-Maskable Interrupt routine software. In this manner the processor can be halted, the required information saved on the stack to allow a return, and then vectored to a new set of instructions. Upon completion of these instructions, the processor can return and pick up where it left off, until interrupted again.

When an Interrupt Request occurs, provided the Interrupt Disable Bit is clear, the current instruction is again completed, the Disable Bit is set, and the Program Counter and Status Register saved on the stack. The processor then branches to a different address in memory ($A678) for the starting address of the Interrupt Request Service Routine, and subsequently jumps to that indicated portion of memory for execution of the instructions which deal with Interrupt Requests.

After the interrupt has been serviced, whether NMI or IRQ, the processor returns to the location of the next instruction to have been executed had the interrupt not occurred. This is termed a Return from Interrupt (RTI), and is accomplished by pulling the Program Counter and Processor Status Register off the Stack, where they were previously stored. Execution then begins where the processor left off. Therefore, when routines are in progress where timing is critical, such as while waiting for a sonar echo, the Interrupt Disable bit should be set to prevent a lengthy interruption at a critical point, and then cleared upon completion of the routine. Any interrupts which may have occurred while the Disable Bit was set will be serviced as soon as the bit is cleared, as the IRQ line is level sensitive.

The NMI line cannot be ignored by the processor under any conditions, and so NMI service routines should be kept as short as possible where there is a chance they could interfere with other routines in progress. The Non-Maskable Interrupt is therefore used to keep track of the time, incrementing the hours, minutes, and seconds registers as appropriate. Since it cannot be masked out, and has priority over Interrupt Requests, the accuracy of these registers is assured, and service time is kept at a minimum. These interrupts are caused by hardware circuitry located on the Clock/Calendar Board, as discussed elsewhere in the text.

The IRQ line connected to pin 4 of the 6502 CPU can be pulled low by either of the three 6522 Versatile Interface Adapters, or the 6532 RAM I/O Timer (RIOT). Only one of these devices is used for interrupt handling on the prototype robot, however, namely 6522-2. Each 6522 VIA has seven potential interrupt sources: four control lines, two timers, and one shift register. Only two sources are used: interrupts generated by Data Selector A (IRQ-A) are fed in on control line CA2 via AA Connector pin 4, and interrupts generated by Data Selector B (IRQ-B) are fed in on control line CB-2 via AA Connector pin 5. These are referred to as Interrupt Channel A and Interrupt Channel B, respectively. Data Selector C has no interrupt capability.

Each Data Selector has the capability to handle sixteen different inputs, any one of which can generate an interrupt. Diode jumpbers are installed on each selector input where an interrupt is required, as shown in Figure 6, Section I-C. Data Selector A utilizes all sixteen inputs to monitor the impact sensor switches and the near-infrared proximity detectors used for collision avoidance. All inputs are jumpered to cause an interrupt, but the four infrared inputs can be disabled at the infrared driver board so as to be ignored, by calling Subroutine I/Rdis. Selector A interrupts can be collectively disabled, without affecting Selector A inputs, by calling Subroutine IRQAdis, and enabled with Subroutine IRQAen. Selector A is polled by Subroutine IRQA once an interrupt is detected. Data Selector B is used to monitor internal circuitry check points, and not all inputs cause interrupts. Examples of those that do are the low battery condition, analog to digital overflow, smoke alarm, fire alarm, and power distribution bus inputs. Selector B is polled by Subroutine IRQB, for branching to the appropriate service routine after returning from interrupt. (See sections on Behavior Selection and Alarms.)

The Interrupt Request Service Routine first saves all primary registers on the stack. Next the Return Register is cleared, and the drive is stopped with the old command stored for later use. Subroutine IRQA is called for polling Data Selector A, with Q (the IRQ index counter) initialized to start with input 4. The structure of Subroutine IRQA is such that it will not return until all inputs on Selector A are low (see Figure 10). Thus the individual service subroutines called by Subroutine IRQA must clear the cause of the interrupt, or no return is possible.

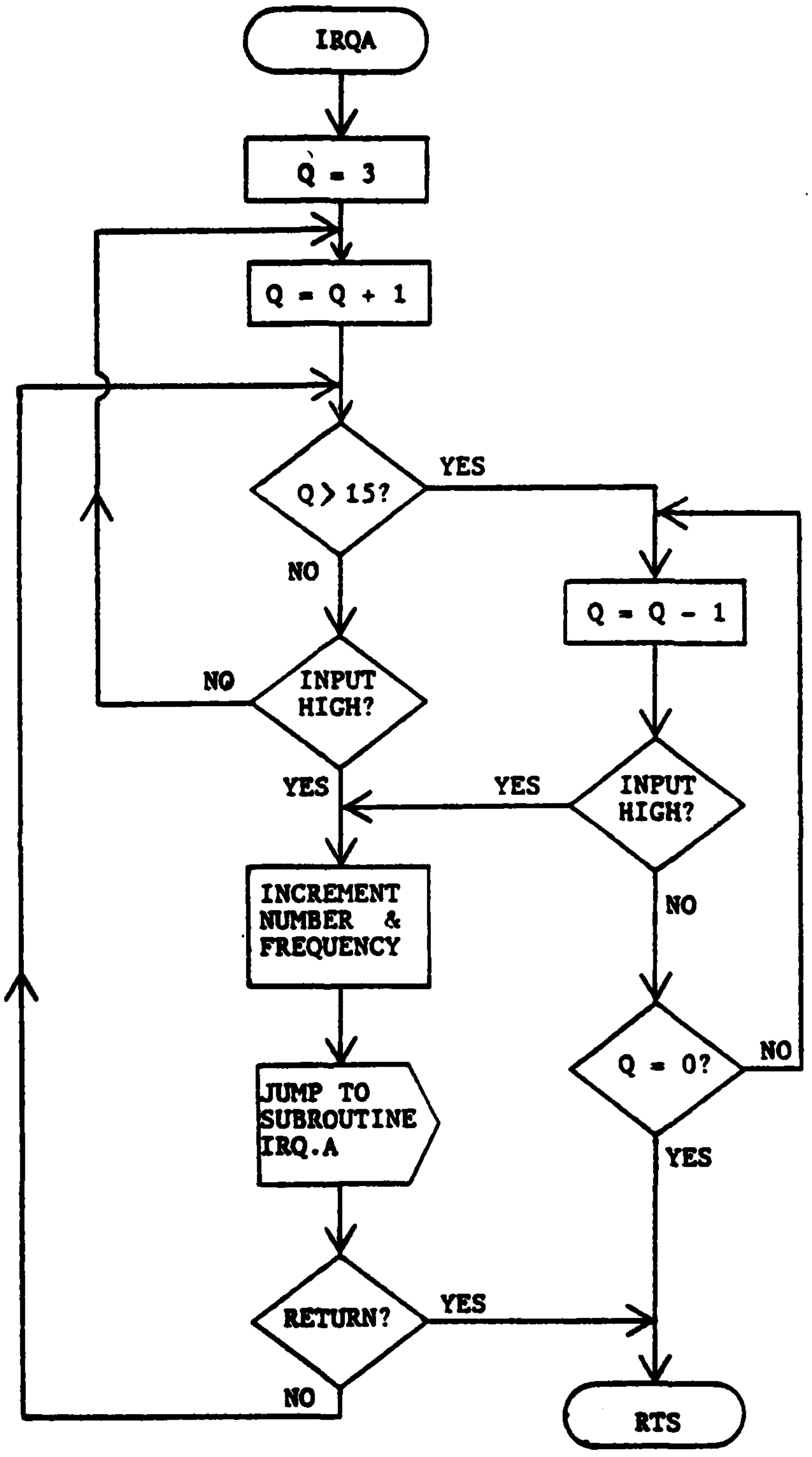


Figure 10. Flowchart for Subroutine IRQA

Since Selector A reads collision related inputs generated by either tactile sensors or the close-in proximity detectors, the service subroutines called must react by energizing the drive wheel to move the platform away from the object sensed. When all inputs have been restored to a low state, Subroutine IRQA returns to the Interrupt Request Routine, which then calls Subroutine IRQB. Subroutine IRQB polls all inputs on Selector B in a similar fashion before returning, after which the Interrupt Request Routine restores the old drive commands, recalls the primary registers, and returns from interrupt.

In addition to effecting drive motor responses within itself, the IRQ Service Routine also causes actions to be performed after return by setting certain registers before returning from interrupt. Register Return will cause a return from interrupt even though all inputs on Selector A are not low, if set by Subroutine IRQA. Register Homing, if set, will cause Skirt to be activated during docking with the charger. A side impact at close range to the charging station will set Register Realign, causing Subroutine Align to be called when docking. These registers are predominantly addressed by Interrupt Channel A related software concerned with collision avoidance. As further examples, Register Exit can be set to ensure termination of a Behavior Subroutine in progress, and Register Next can be set to pick the subsequent Behavior Subroutine. This technique is used to process and react to the alarm conditions generated by Interrupt Channel B.

II. THE AUTONOMOUS ROBOT SENTRY

With the daily passage of time industry and the general public are becoming more acutely aware of the emergence and potential of the robot. From the vague definition of "machines that think" there are evolving a multitude of very sophisticated and practically proven assembly line robots, and a backward glance at progress over just the past few years yields awesome visions of what the future may soon hold. With speech synthesis now an easily achieved reality, and speech recognition showing much promise of the same, the potential of the robot to serve as a valuable assistant to man in areas beyond the scope of a controlled assembly line environment **is** becoming more and more apparent. The relatively new field of artificial intelligence is now the subject of extensive and rewarding research. Robots which can function on their own, converse with humans, and move about performing tasks in toxic or radioactive areas otherwise inaccessible, are no longer merely conjecture but reality, and their numbers can be expected to swell at a staggering rate.

In the development of an autonomous system, the designer must address several fundamental areas after the initial decisions are made with regard to CPU selection, drive mechanisms, and structural framework. Of primary importance should be the question of control. Very few systems will be brought up on line with no human intervention whatsoever, particularly in the initial stages of development and up through preliminary tests of the resultant product. Exactly how much human interaction is needed and how it should be implemented are questions that need to be resolved early.

An intelligent mobile platform obviously needs to be provided with a real time reference to assist in the performance of its behavior routines, and various means are available through which both time and date information can be made available to the CPU.

As a prototype begins to take shape from these meager beginnings the need for automatic replenishment of its energy store quickly surfaces. As battery power is currently viewed as the most practical supply for indoor operation, this usually requires the development of a tracking system to locate an available battery charging station, and a means of effecting the proper connection.

Since the platform must move about in the performance of its duties and, to ensure survival, an ability to navigate and avoid collisions with surrounding objects is crucial. This is perhaps one of the larger areas of concern, and involves careful selection and placement of sensors as well as effective data utilization within the software.

Once the capability to maneuver safely has been added, the refinement of the machine's ability to accomplish its overall objective becomes the dominant issue. In the prototype robot ROBART this objective is to serve as an autonomous sentry in a standard home environment. Therefore considerable attention must be paid to methods of intrusion detection, as well as means to indicate the presence of smoke, fire, toxic gas, flooding, and other unwanted conditions, and the development of software routines to deal with each situation.

Up until now the software associated with the concerns discussed could conceivably consist of reactionary subroutines which detect a condition and respond accordingly after polling a few appropriate inputs which dictate the response. As complexity grows, however, this means of control becomes severely limited, and unable to deal with unforeseen problems the machine is likely to encounter. A hardware and software structure for controlling the robot's actual motions must be developed that can support a higher level program tasked with determining what the machine's responses should actually be. At that point the device is ready to be provided with its own artificial, or machine, intelligence.

In the following sections each of these fundamental areas will be addressed as implemented on the prototype. The development of the operator/machine interface is explained, followed by a description of the system's real time reference. The sophistication of the machine increases as the ability to locate and connect with a free standing recharging station is added, as well as collision avoidance capability. The discussion concludes after examining the prototype's means of intrusion detection and the manner in which behavior routines are requested and subsequently performed.

A. OPERATOR CONTROL

Control shift between the CPU and the system operator is handled by the Subroutine Control, which when called checks the external input switches 1 through 4. If any of these switches is up, the CPU will initiate procedures to turn over control to the operator, otherwise the subroutine returns with no action taken.

The control shift procedure must make provision for several things:

1) An effective yet simple method for determining if the operator desires to take control, or return control to the CPU.

2) The passing of control at the appropriate points in the program flow, so as not to disrupt the completion of events which should be terminated before other events are begun.

3) An efficient method to determine which routine the operator wants to perform or request.

4) A means for inputting parameters or data needed to execute or clarify the chosen routine.

If control is requested, Subroutine Control will perform all these functions, interacting with the operator through voice instructions. This makes the process fairly easy to follow, and requests are handled in a clear and orderly fashion. The CPU will first set the IRQ interrupt disable flag, stop the drive wheel, store the old drive command in

Register Dri.com, indicate that control is being passed to the operator, and then request the 'Service Select Entry'. This is a four bit control code entered via the external input switches. The operator will be instructed to set the switches and then to press the ENTER button.

Pressing ENTER triggers a 555 monostable multivibrator located immediately behind the switch panel, and the ENTER LED goes high. The 555 effectively debounces the push-botton, holds the ENTER signal high long enough to be read (two seconds), and then resets. Meanwhile, the CPU calls Subroutine Ent.chk (entry check) to read the ENTER signal on PA7 of 6522-3. This subroutine assigns the value of hex 10 to Register Count ($06). It then reads PA7 every half second, looking for a high, and decrementing Count each time. If PA7 goes high, or the Count register reaches zero, the subroutine returns. On return, if Count equals zero, there was no entry, and the request to enter is repeated. If Count is non-zero the CPU then reads the entry switches, which by this time have settled and thus require no de-bouncing. The four bit code just read determines which service the operator is requesting, and is stored for the time being in Register Ser.cd ($09). The CPU then requests the 'Control Code Entry', which is inputted in a similar fashion and stored in Register Con.cd ($10). The Sub-routine Op.exec (operation execute) is then called to perform the required service based on the contents of Register Ser.cd, subject to the constraints represented by the contents of Register Con.cd.

There are fifteen Service Routines which the operator can request, and on the prototype robot these are primarily used for trouble shooting the system or modifying its behavior. As an example, Service Routine Number 1 can be called to cycle Data Selector A and report by voice the input state as seen by the CPU. The 'Control Entry' (contents of Register Con.cd) entered by the operator specifies the starting input for the test. Upon completion, the CPU will instruct the operator to enter any desired changes in the 'Control Entry'. The 'Control Entry' will then be read and the original requested service again performed subject to the new constraints. This process will continue until the control code of zero is entered, which signals passing control back to the CPU. The control shift is announced, and Subroutine Pl.dri (pull drive) is called which fetches the old drive command from Register Dr.com ($11) where it was saved, IRQ interrupts are re-enabled, and the CPU resumes where it left off before shifting to operator control.

B. REAL TIME CLOCK

The real time clock used by the system is implemented through software but derives its timing pulses from hardware circuitry located on the clock calendar board in the head. Here a National Semiconductor Clock/Temperature module is driven by a quartz oscillator via a seventeen stage divider (MM5369) which produces a precise 60 Hertz reference square wave. When its display is activated by the 3 volt DC supply, this module will display the time in hours and minutes on its own seven segment display. The temperature in either celsius or Fahrenheit can also be displayed, as well as seconds and alarm setting. The module provides only one digital output: AM/PM. This signal is used to drive a binary counter which counts from zero to six, and then resets. This count is displayed on a single seven segment LED located on the Clock/ Calendar Board, and represents the day of the week (Sunday = 0). The count is also parallel fed to Data Selector C for use by the CPU. The CPU has no direct way of reading the internal clock module registers to ascertain the hours, minutes, and seconds, however.

To get around this problem, the 60 Hertz reference is fed to a divide by 60 counter, to produce a one Hertz square wave. This in turn is applied to the Non-Maskable Interrupt input on the CPU, and generates an interrupt once a second. The NMI Routine then increments the Seconds, Minutes, and Hours Registers in the appropriate fashion to duplicate the internal registers in the clock module. If the CPU registers are first set to the time on the module display at system start up, the NMI routine will ensure that the registers hold the accurate time, available for subsequent use by the software. The Non-Maskable Interrupt routine also clears Register IRQ.fre, which is used to keep track of the number of maskable interrupts per minute, each time Register Minute is incremented. This provides a useful piece of information for the collision avoidance routines as discussed later.

The one Hertz square wave is essentially symmetrical with a 50 percent duty cycle, and its state (high or low) can be read by the CPU over Data Selector C, as a more or less random variable for logical decisions throughout the software. An example is the random deletion of the word 'is' in the voice output generated by Subroutine Time to avoid constant repetition of the same phrase as discussed below.

Subroutine Time is a speech synthesis program which outputs the time in hours and minutes when called. The design intent was for speech output on the hour and half hour, without interruption of critical routines which may have been in progress. Speech output is of the form: "The correct time is --- hours and --- minutes". A second consideration was cancellation of voice output after such time as the household retired for the night. These two objectives are handled by Subroutine Clock. First, the NMI Routine, upon detection of either a 'zero' or a 'thirty' in the Minute Register ($01) will assign the appropriate speech synthesis address for that quantity to Register Timeflag ($04) (i.e., zero will be $1f, thirty will be $15, on VOX1). The NMI Routine takes no other action, besides periodically clearing Register IRQ.fre, and keeping an accurate count in the time registers. Thus, the time required to service a NMI interrupt is kept as short as possible.

Subroutine Clock can now be called at appropriate times in the main program flow, when voice output of the time does not interfere with other routines. This subroutine first checks the Timeflag Register, and returns with no action taken if the register contents are hex zero ($00). If either of the aforementioned speech synthesis addresses is encountered, however, it is time to voice output the hours and minutes. Data Selector B is first read for ambient light conditions, and if the room is dark, no output takes place, and the subroutine clears the Timeflag Register and then returns.

If the room is not dark, Subroutine Clock then calls Subroutine Time which outputs the hours and minutes. The speech synthesis address for hours is read directly from the Hour Register ($00). Since this register value is in decimal, a conversion to hex must be made for ten, eleven, and twelve o'clock, for the speech synthesis Subroutine Voxld to function correctly. The address for the tens of minutes is read directly from the Timeflag register, where it was stored by the NMI Routine. Units, up to nine, are read directly from the lower four bits of Register Minute. Upon return to Subroutine Clock, the Timeflag Register is then cleared, and Clock returns to the main program.

Subroutine Clock can be called from any appropriate point in the main program flow or in selected subroutines. Subroutine Delay2c, used to create delays between routines, calls Clock about once every half second throughout the delay period. The end result is that the NMI Routine is freed of all constraints addressed by Clock and Time, and the time registers will remain accurate. Additionally, speech output is not triggered directly by the interrupt, and therefore will not occur in the middle of a program routine already in progress. In most cases, the actual delay thus created between setting the Timeflag Register and the actual recognition and speech output of the time will not be more than a few seconds. Since seconds are not output, no error will be noticed between the voice output and that of the LED Display on the clock module in the robot's head.

As a further refinement, Subroutine Time checks the actual seconds count in the Seconds Register ($02), and if more than 15 seconds have elapsed, the word 'correct' is deleted from the speech output. This also helps reduce constant repetition of the same phrase every 30 minutes. However, if ten or more minutes have elapsed since the setting of Register Timeflag by the Non-Maskable Interrupt routine, the time is not announced, and Subroutine Time clears Register Timeflag and then returns.

C. AUTOMATIC RECHARGING CAPABILITY

For a machine of this type to be useful operating in environments hazardous to humans, it not only must be immune to the hazards, but also able to support itself to a large degree. Of prime consideration will be the ability to recharge itself when a low battery condition is imminent, thus freeing itself of the hindrance of an attached cable to ensure adequate power levels. Detection of the need to recharge is easily implemented, and the problem becomes one of how to locate and connect to a centrally located charging station (or stations). An extremely reliable process is required, one which is not rendered ineffective by unforeseen obstacles or changes in the robot's environment, and whose complexity does not overshadow the primary function for which the device is employed. A self-sustaining robot becomes truly an asset when properly equipped with the required hardware and software to perform needed tasks in isolated reactor compartments, on unmanned offshore oil platforms, and at contamination sites, to name but a few examples.

The requirements of simplicity and reliability effectively rule out the standard approach to the problem: trying to align a special plug on the robot with a mating receptacle on the charger. While this can be done, it requires rather complicated hardware as well as software, and is by nature susceptible to complications. What is needed is a method of locating the charger and making contact that is independent of the direction of approach, requires no complicated alignment procedures, and provides a good electrical connection every time. While this sounds like a tall order, a very effective solution can be realized quite simply.

1. Automatic Scan and Track System

Location of the charging station by the robot can be accomplished by any of several methods, but the requirement for high resolution with short range needs makes an optical tracking system a prudent choice. A visual homing beacon can be attached to the charging station in such a fashion as to be 'detectable' when activated by the robot via a radio link, as depicted in Figure 11. This beacon can consist of an ordinary incandescent lamp situated at the same height as the photocell detector sensors on the robot or, better, a pulsed infrared source that can be distinguished through its unique repetition frequency by a tone decoder in the detection circuitry. The pulsed source eliminates the need for a verification step in the software to allow the robot to ascertain it is indeed looking at the correct light if an incandescent beacon is used. Since its light is invisible, a near-infrared beacon could be continuously energized, and so the radio link could be eliminated. Having located the charger, by whatever means, the robot must be capable of tracking the beacon while moving towards it.

In the prototype robot tracking is implemented as an integral part of the hardware circuitry which controls the

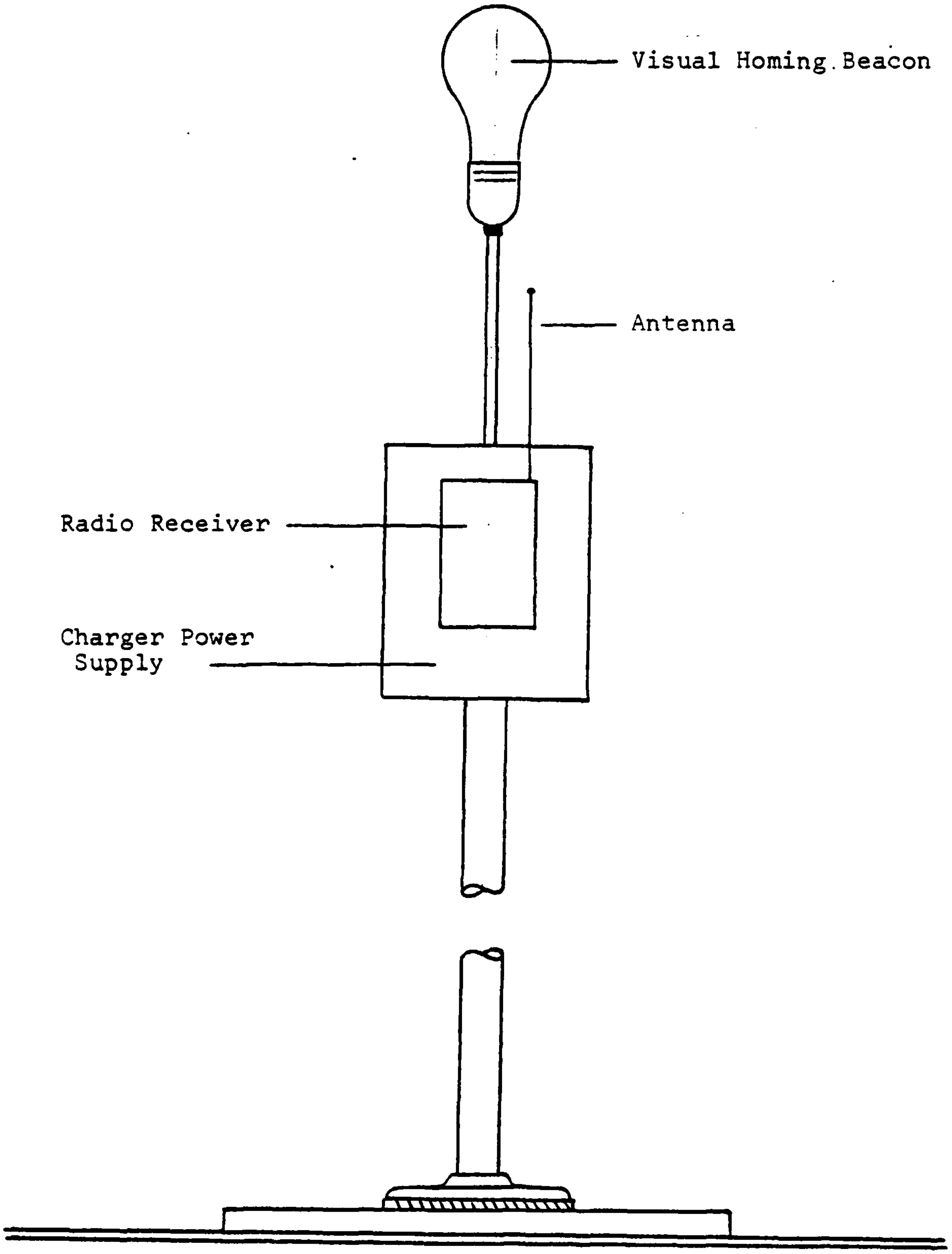


Figure 11. Free Standing Recharging Station. Beacon at top of pole is controlled by robot via radio link.

head position, located primarily on the Optical Board and Interface Board Number 5. The head positioning motor is directly controlled by the Optical Board, and Interface Board Number 5 evaluates the CPU commands in conjunction with other inputs to determine the control mode. This circuitry (Figure 12) allows for three different schemes to control the motion or position of the head, referred to as: 1) scan mode, 2) track mode, 3) position mode. The controlling mode is selected by setting the appropriate levels (high or low) on the Scan Enable, Track Enable, and Position Enable lines, and special subroutines are provided to do this. A brief explanation of these three control modes follows.

In the scan mode the head sweeps back and forth between full right and full left positions, 100 degrees either side of centerline. This action is controlled by the scan flip-flop on the Optical Board inside the robot's head. This flip-flop is set and reset by limit switches at both extremities of head travel, causing the motor to reverse direction each time, and the head to scan the other way. This mode is selected by Subroutine Scan which sets the Scan Enable line high, and the other two control lines low. When the Scan Enable line goes low, the head will be immobilized, provided the Position Enable line is also low. All three lines are set low by Subroutine Scanoff.

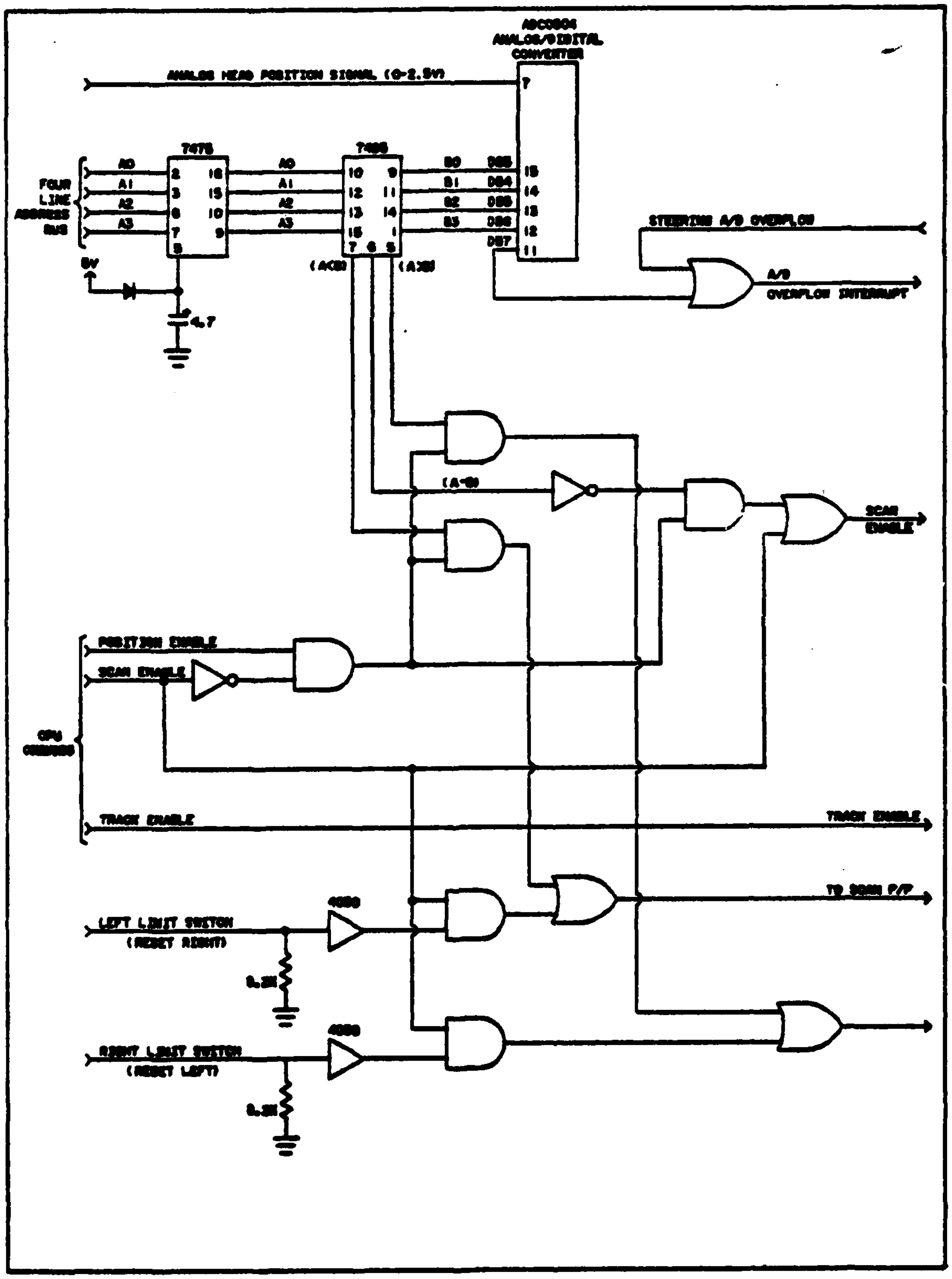


Figure 12. Interface Board Number 5. Determines control mode for Optical Board as requested by CPU.

If the Position Enable line is high, however, the head will seek the position stored by the CPU in the 7475 four-bit latch on Interface Board Number 5. A 7485 four-bit comparator compares the command stored in the latch with the output of the head position A/D convertor, and positions the head accordingly, in the same way similar circuitry on Interface Board Number 2 positions the drive wheel. This head positioning circuitry is automatically gated out whenever the Scan Enable line goes high. The latch is loaded from the interface four line address bus, and enabled by PB7 on 6522-2 (orb2). These operations are automatically performed by Subroutine Latch, which takes the desired position command from the Y register. Thus the head can be made to center itself after a scan operation, or to seek any of sixteen fixed positions on command.

If both Track Enable and Scan Enable are high with Position Enable low, then the system functions in the tracking mode, and the head looks for, locks on to, and follows the homing beacon situated on the recharging tower. (The system will actually track any bright light source, and it is up to the software to ensure that this source is the beacon.) The tracking sensors consist of three photocells arranged in a horizontal array on the head, each with a half-inch diameter collimating pick up tube 12 inches long. The tubes are arranged in a diverging configuration (5 degrees between tube axis centerlines), with the center tube parallel to the forward axis of the head. The three tubes are shown at beacon level in Figure 13.

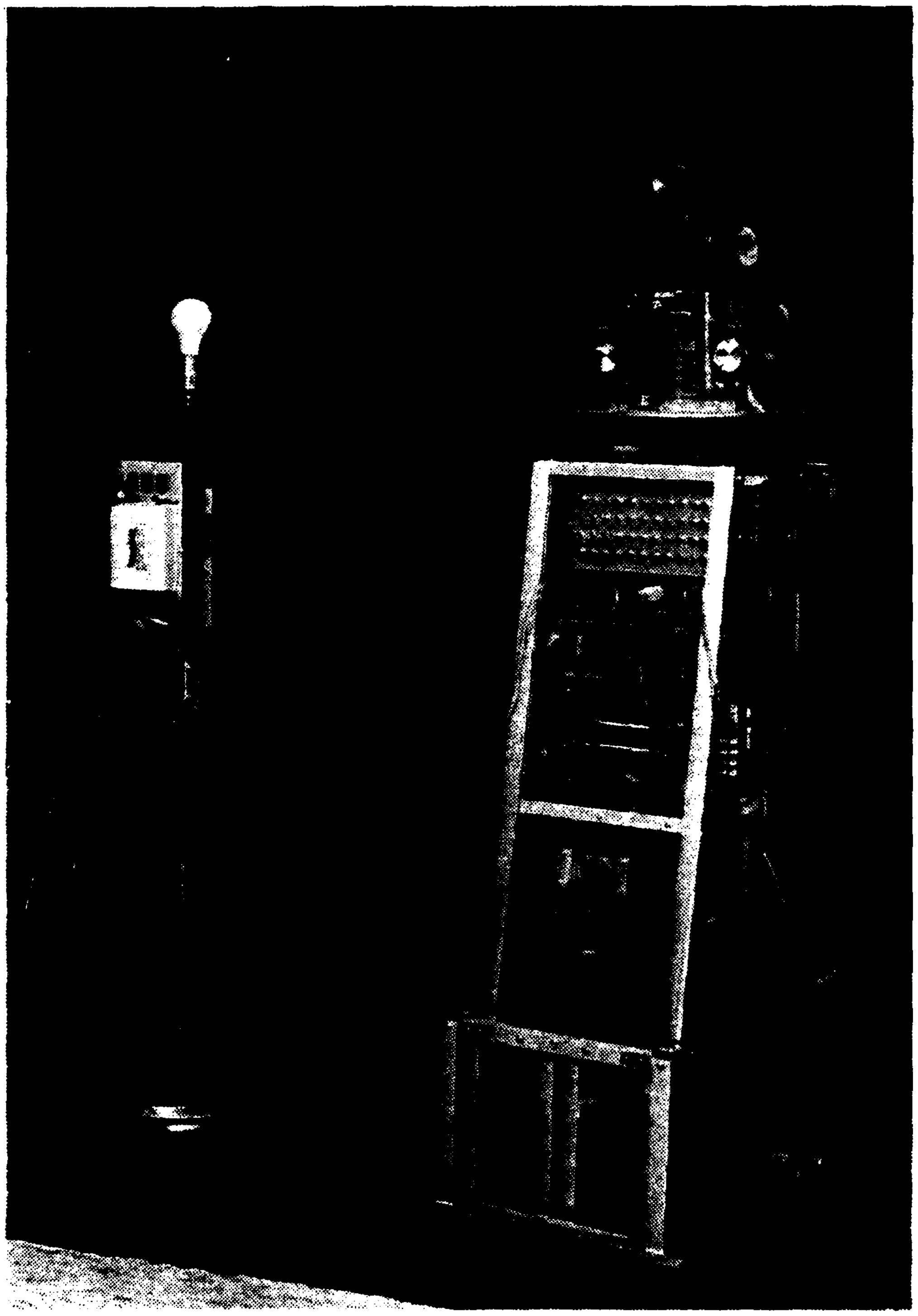


Figure 13. Photo of Robot and Recharging Station. Collimating pickup tubes for photocell array are located at beacon height on front of head assembly, shown here turned slightly towards beacon.

The Optical Board provides three digital outputs pertinent to the tracking process: 1) Target Output, 2) Point Source Output, 3) Range Output (see Figure 14). The Optical Board Target Output goes high if any of the three comparators associated with the photocell array goes high, while the Point Source Output reflects the status of the center photocell comparator only. The Range Output indicates relative distance from the homing beacon, as discussed later. Thus the CPU communicates with the hardware tasked with tracking the beacon with a total of six lines: three control and three output.

The tracking process is initiated automatically by a low battery condition through alteration of the Behavior Selection procedure in such a way as to terminate the routine in progress. When the battery condition goes low and remains below the set point for more than 5 seconds, a flip-flop on the Monitor Board changes state and triggers an interrupt. The IRQ routine which handles the Channel B interrupts disables the low battery interrupt, and sets Register Next to select the docking routine (see section on Behavior Selection and Alarms). The transmitter which turns on the homing beacon is activated, and the CPU enables the Automatic Scan and Tracking circuitry, while sending the Position Enable line low. The head begins to scan left and right, seeking a point source of light of sufficient intensity to trigger the photocell comparators. This action continues as long as Scan Enable and Track Enable are held high by the CPU, and no

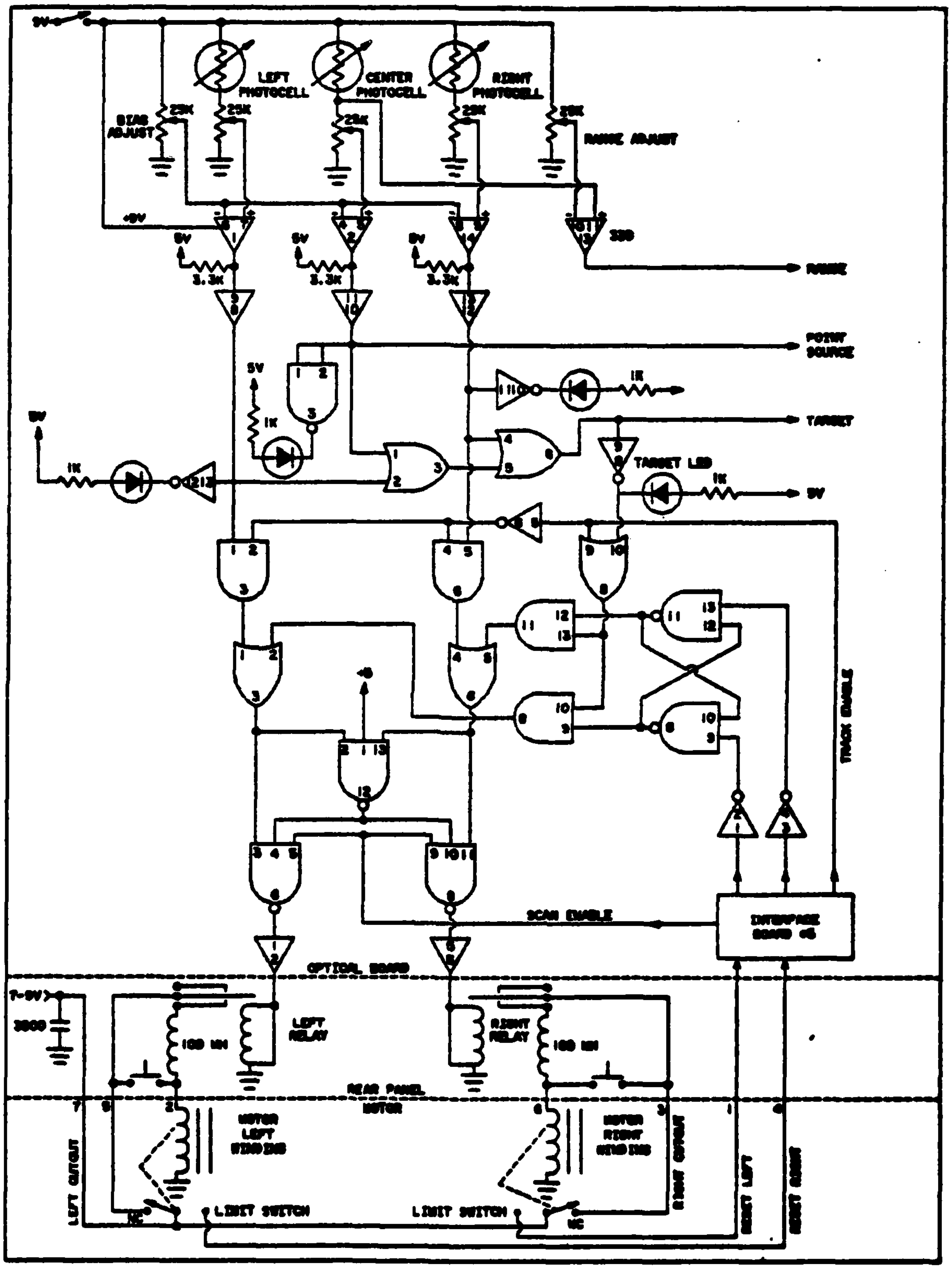


Figure 14. Optical Board Circuitry Schematic. Controls head position in one of three modes as determined by Interface Board Number 5 (Figure 12).

light source is detected. If any of the three optical comparators goes high, indicating acquisition, the scan flip-flop is gated out automatically, and the tracking inputs take over control of the head positioning motor.

The tracking inputs to the optical board circuitry come from the left and right photocells in the array. Their respective comparator outputs (Figure 14) indicate a greater light intensity either side of center, referenced to the center photocell output. The appropriate positioning motor winding is energized, and the head turns to regain maximum intensity at the center photocell, thus tracking the source. (If by chance both left and right photocells showed intensities greater than center, both inputs are gated out and the head remains motionless.) All this happens only if at least one of the photocell outputs is above the adjustable set point provided by the Background Light Bias Circuitry on the Optical Board, otherwise the system reverts to the scan mode and searches for a bright light source. Any comparator output signalling intensity above the set point gates out the automatic scan, and the tracking inputs take over. When the array outputs indicate the head is correctly positioned (pointing at the source), the motor windings will be de-energized.

The three collimating tubes limit the photocell fields of view to relatively small regions, and the beacon is situated at just the right height so as to be centered vertically within the detection zone when one of these tubes is pointed at the recharging station. This results in a relatively high signal to noise ratio as the head scans back and forth in search of the beacon, as shown in the stripchart recording of Figure 15. The sharp peaks resulting from beacon acquisition readily stand out over the signal produced by ambient lighting during the sweep.

The Background Light Bias circuitry is tasked with providing the comparators with a reference voltage above which photocell output is probably due to a point source of sufficient intensity to possibly be the homing beacon. The initial design provided for a bias potentiometer to manually set the reference level. This proved inadequate due to oversensitivity in close to the beacon. If the bias threshold was set low enough to allow detection of the beacon at long range, then the system saturated in close and all comparators went high when the pickup tubes were pointed in the general direction of the light. What was needed was a means of reducing the sensitivity from that needed for long range detection, as the robot approached the recharging station, to a level yielding good bearing resolution in close, where accuracy became critical.

The circuitry employed on the prototype basically provides for two manually set reference points, one to allow sufficient sensitivity for long range detection and a second with greatly reduced sensitivity for short range use, to

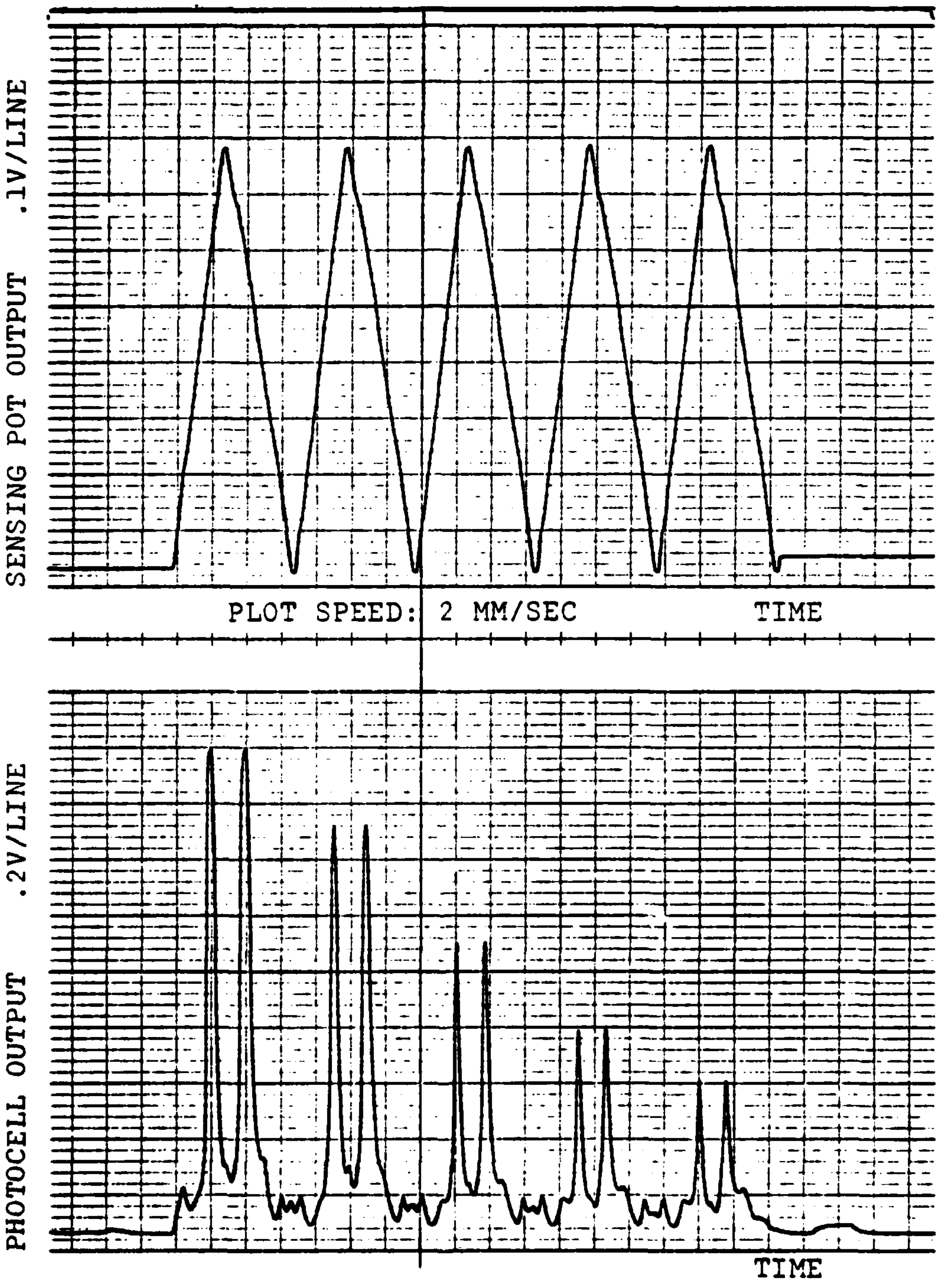


Figure 15. Photocell Output During Automatic Scan Sequence. Upper plot represents analog signal from head position sensing potentiometer. Lower plot indicates center photocell output, with peaks resulting from beacon acquisition at ranges of 3, 6, 9, 12, and 15 feet.

force the tracking system to point the head directly at the light. This provides much more accurate bearings to the beacon for the purpose of docking with the recharger. A fourth comparator monitors the center photocell voltage output and effects the changeover when its output signals the robot is within three feet of the beacon, based solely on the perceived light intensity (Figure 16). This comparator output is also made available to the CPU (Range Output), and is used in many software routines to determine relative range (near or far) to the charging station.

Once the CPU ascertains that the head has a lockon (the Target Output goes high), it must interrogate the source to verify that it is indeed the beacon. Verification is accomplished by setting the Scan Enable low, and then turning off the beacon with the radio transmitter, observing to see if the Target Output line went low. An incorrect source will keep the Target Output high. The Scan Enable is set low so the head will not start scanning again if the source does go out, and the Position Enable must also be low so the head will not seek the position dictated by the contents of the 7475 latch when the Scan is set low. If the source is not the beacon, the CPU sends the Track Enable line low so the source will be ignored, sends Scan Enable high to reinitiate the scan, and waits as the head turns for the incorrect source to clear (Target Output goes low). As soon as this happens, Track Enable is sent high again, and a new source

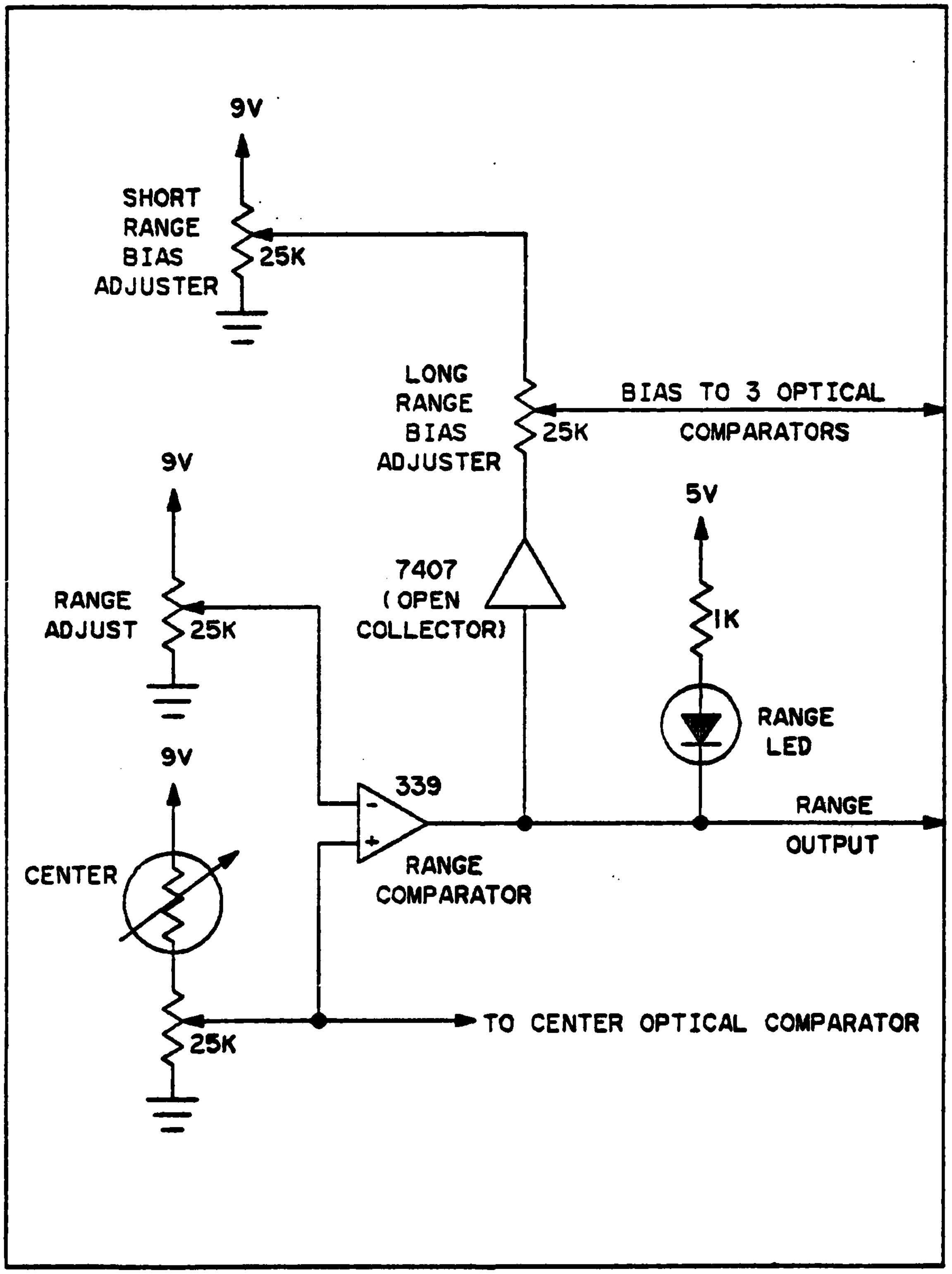


Figure 16. Background Light Bias Circuitry Schematic. Range Output is used to indicate relative distance to beacon (near or far). Bias to three optical comparators changes as robot approaches recharger, thus decreasing system sensitivity and preventing saturation in close.

is sought. The process repeats until the correct source is found, until three incorrect sources have been interrogated, or until a specified time limit has elapsed. The latter two indicate that the beacon is not in the immediate scan field (first and fourth quadrants relative to centerline), and the robot will perform a Y-turn and check the other side.

Once the beacon has been located and its bearing and relative range announced through speech synthesis, the robot begins to home in on the recharger. The CPU repeatedly reads the head position, representative of the bearing to the charger, and sends an identical command to the steering motor via Interface Board Number 2. As the robot turns, the head automatically tracks the source, and the relative bearing to the beacon decreases. The CPU therefore is subsequently decreasing the turn angle, until eventually the source is directly in front of the platform, and the drive wheel is centered.

A four-bit analog to digital conversion of the potentiometer output voltage which represents the head position yields sixteen possible head positions, evenly distributed around the 200 degree arc of scan. This produces cone shaped sectors every 12.5 degrees, and this resolution is far too coarse to precisely fix the beacon position as required for successful docking with the charger. Therefore the potentiometer output was conditioned to yield the modified output shown in Figure 17, which in effect compresses the resolution into the center of potentiometer travel, where the head is

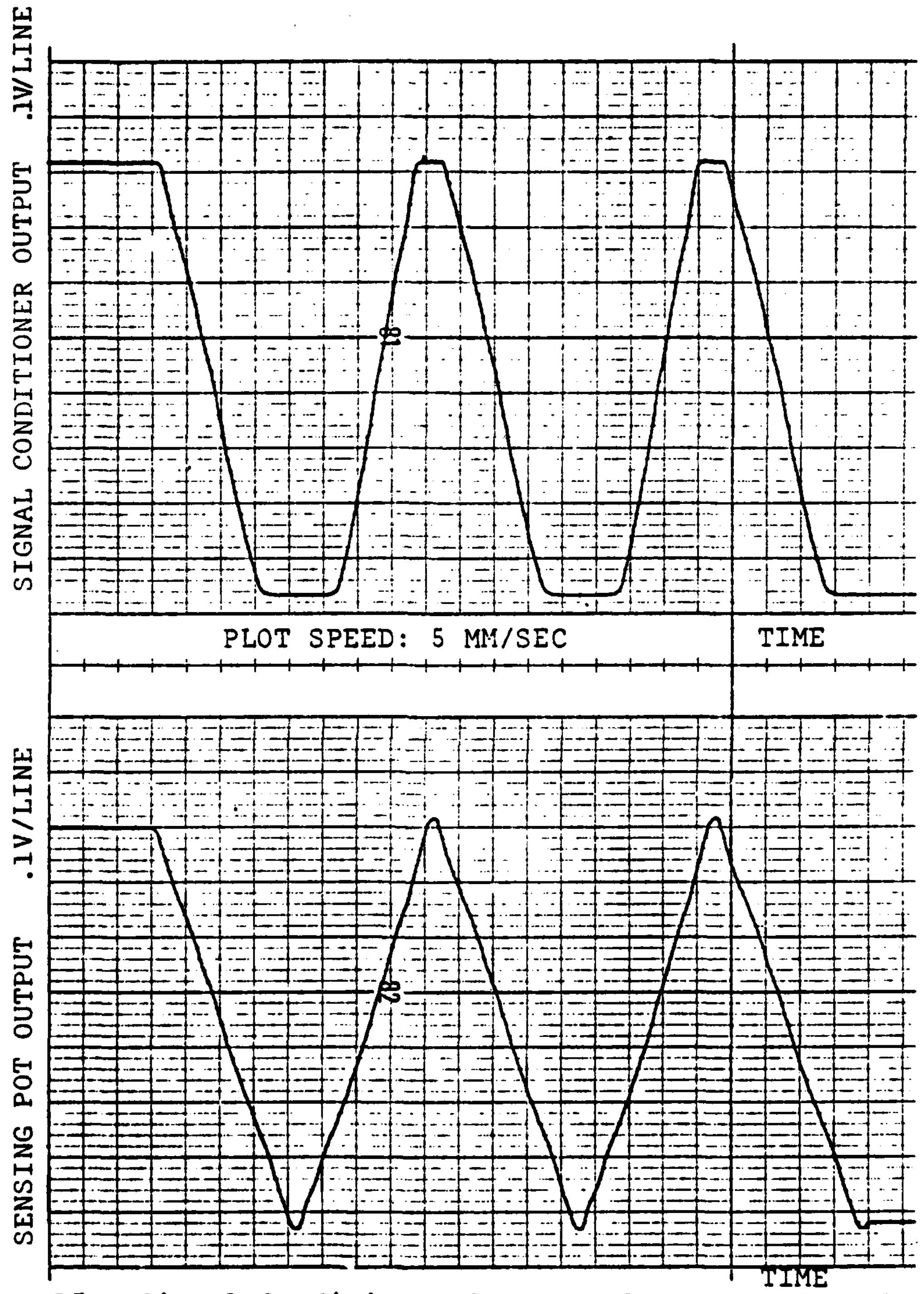


Figure 17. Signal Conditioner Output Voltage. Upper plot represents conditioned analog signal for head position as seen by A/D convertor. Lower plot indicates actual sensing potentiometer output before conditioning. Steeper slope in upper plot results in greater resloution of head position in center region of scan.

pointing directly ahead. The output of the signal conditioner remains at zero volts until the sensing potentiometer voltage exceeds 1.0 volts, at which time it increases linearly with the input voltage from the pot. The output voltage is clipped at 4.0 volts, and remains constant as the input voltage continues to increase. This output voltage is then attenuated to exactly 2.5 volts at maximum output and applied to the A/D convertor, which produces a binary output ranging from 0 to 15 (0000 to 1111).

As a result of this conditioning any head position from extreme right to 45 degrees right is seen as position 0, and similarly any position from 45 degrees left to full left is read as position 15. The other 14 sectors are evenly distributed over the remaining 90 degrees, 45 degrees either side of centerline. This greatly improves the resolution in the center of the scan where accuracy is needed for homing on the charger.

2. Docking System

The task of making contact with the recharger can be greatly simplified if the circuitry involved is restricted to that associated with the battery voltage, 12 volts for example, as opposed to the more dangerous 117 volts of a normal AC distribution system. The lower voltage level allows for contact surfaces to be exposed with no electrical shock hazard. If these exposed contact surfaces are made axi-symmetrical with respect to the vertical pole which supports the visual homing beacon, they will present the same target to the mating contacts on the approaching robot, regardless of the direction of approach. Thus the need for a critical alignment to ensure contact is eliminated.

Two connections must made to complete the recharge circuit, and will be referred to as HOT and GND. For reliability the connecting or mating actions of the two should be independent of each other. This can be accomplished by making their respective contact axis lines perpendicular to each other, analogous to the orthogonal cutting directions used to isolate two channels in recording a stereo disk. For example, the contacts associated with the GND leg are brought together by a relative movement in the horizontal plane, whereas those associated with the HOT leg are brought together by a relative movement in the vertical plane. Thus the first set of contacts to meet does not impede the motion required to close the gap at the other set, and chances of a good connection at both points are greatly increased. This being the concept, a slight modification (discussed below) will preserve the required independence while requiring movement in the horizontal direction only, which need be only the inherent motion of the robot as it closes on the recharging beacon.

In the actual construction of a test prototype for this concept, the metal pole supporting the optical homing beacon served as the point of contact for the GND leg, its respective mating surface being the front bumper of the robot chassis (see Figure 18). This front bumper was spring loaded to allow it to absorb impact and make possible the closure of numerous microswitches used as sensory inputs for collision detection. This already present spring action serves to keep the aluminum bumper in tight contact with the vertical pole once the two come together, compressing the springs. Once the bumper springs are thus compressed, horizontal motion of the robot must be halted. This condition is indicated to the controlling microprocessor by closure of the contact sensing microswitches activated by the front bumper, in conjunction with the signal indicating that recharging power is sensed on board.

The connection for the HOT leg of the recharge circuit is made through the mating of a circular aluminum plate at the base of the beacon tower and a set of spring probes attached to the front drive wheel support cage on the robot. The aluminum plate is electrically insulated from the vertical pole which serves as the GND connection by a plexiglass insulator between the plate and the half-inch pipe flange into which the upright pole is screwed. The spring probes which mate with the circular plate are vertically oriented in such a way as to extend downward from a small box situated immediately behind the front bumper. As the bumper passes over the plate moving toward the pole, the spring probes are brought into contact with the aluminum plate,

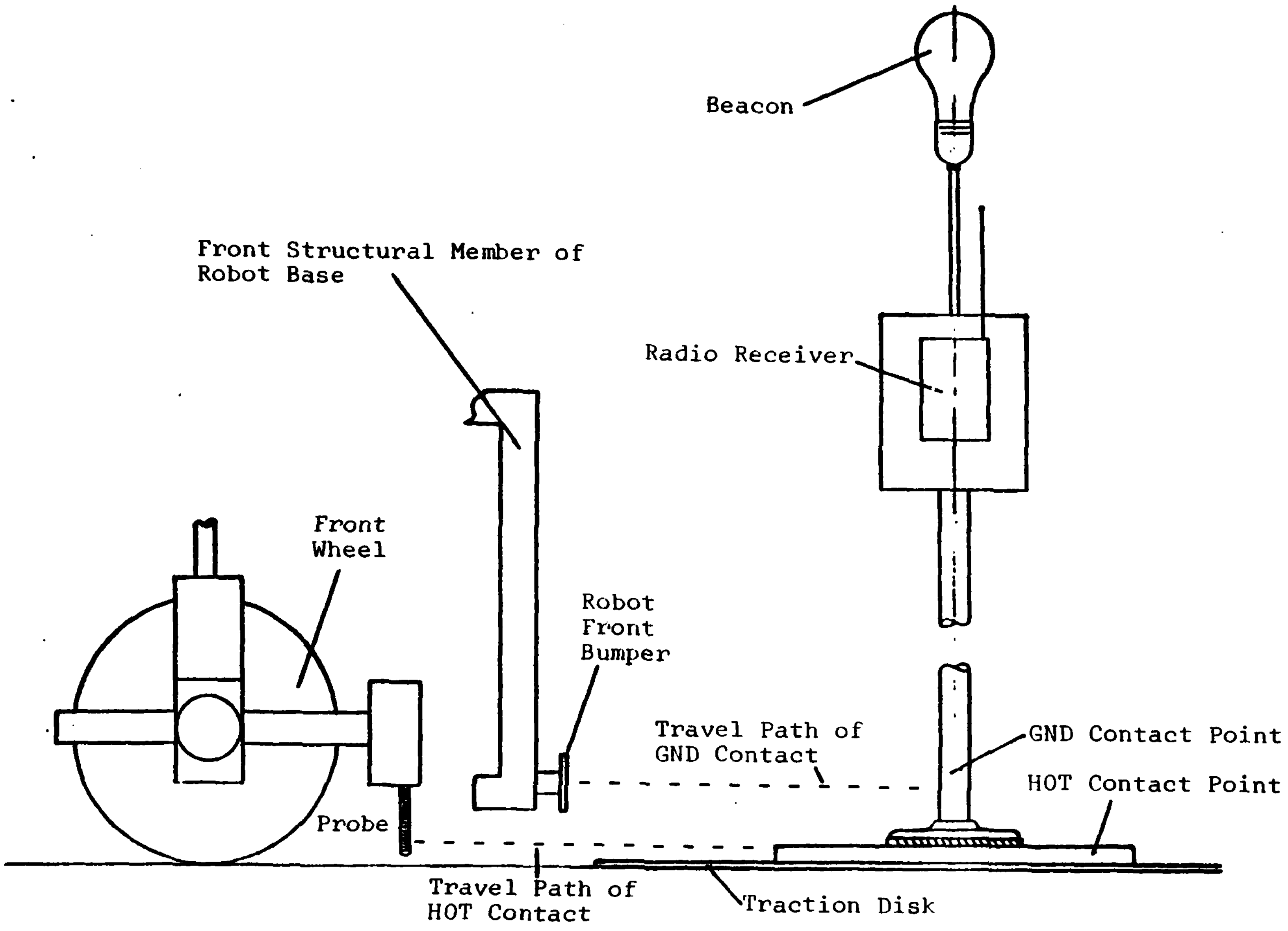


Figure 18. Detail of Robot Approaching Recharging Station

and contact is maintained as the motion continues toward bumper impact. As soon as the bumper contacts the upright pole, the circuit is completed, and recharge current flows to the battery. An electrical relay is connected across the circuit on the robot so as to de-energize the forward windings of the drive motors when final connection is made. This serves as a backup for the software which also de-energizes the drive wheel when the recharge probe potential goes high with respect to electrical ground.

The configuration of the two necessary contacts for the recharge circuit thus ensures the required independence while requiring motion in only one plane, and the use of multiple spring pick-up probes in the HOT leg ensures a good connection with low current density at the mating surfaces. The software can monitor the probes as well as the battery level while the system is recharging, and should electrical contact be lost, effect the necessary forward motion to re-establish the connection. In extensive testing of the prototype, this has not been necessary to date, primarily because the spring loaded bumper provides the necessary force to keep the surfaces pressed tightly together. The geometry of the configuration is such that the probes will be in contact with the plate as long as the front bumper is touching the vertical pole, and since the front bumper for the prototype is 14 inches wide, considerable margin for error is allowed the tracking system which brings the robot into contact with the beacon tower. Even a very simple and inexpensive photoelectric tracking system homing in on an ordinary 75 watt light bulb produced final impacts within two inches either side of centerline in repeated testing (over 200 computer controlled dockings to date).

3. Recharging System

There are two power supplies associated with the recharging station itself. A relatively low power twelve volt source remains energized at all times, and supplies the receiver and decoder circuitry which activates the beacon on demand from the robot. It also energizes the aluminum base plate through a resistor capacitor network to a peak potential of 16.5 volts, which acts as the 'sensing' voltage for the pickup probe, allowing the microprocessor to know when the recharge circuit has been completed. As soon as the battery has been connected as a load on this supply, this voltage level will drop to just slightly over the battery voltage (around twelve volts). This voltage drop is sensed by detection circuitry on the recharging station, which in turn activates the high power battery charging supply, located just below the receiver board. This second supply furnishes the current required to recharge the battery, and is automatically shut off when the robot disconnects and the load is no longer sensed. The robot can deactivate the beacon via the radio link once initial contact has been made and the recharging process initiated, without affecting the recharger supply.

The robot's battery voltage is monitored by an LM339 comparator, which sets a flip-flop after a five second delay when the voltage falls below an adjustable set point (see Figure 19). The delay is used to ensure the battery voltage was not momentarily pulled low by a stalling drive or steering motor. When the flip-flop changes state, an interrupt is generated which is read by IRQ Channel B. The interrupt routine subsequently gates out the flip-flop output, and selects the docking routine (see section II-F).

The battery voltage is also monitored by an LM3914 Dot Display Driver, which drives 10 LEDs to give a visual indication of the battery charge. The upper LED in this display drives another comparator which subsequently changes state when the battery is fully charged, as indicated by the display. This upper set point is also adjustable (see Figure 19).

This entire concept has been tested throughout various stages of completion over the last twelve months, and is currently operational in a fully computerized robot system featuring automatic battery level monitoring, station location and tracking, and subsequent recharging. The system has proven an extremely reliable and easily implemented solution to a problem that must be dealt with if computer controlled robots are to achieve the degree of freedom necessary to perform tasks in isolated or dangerous environments.

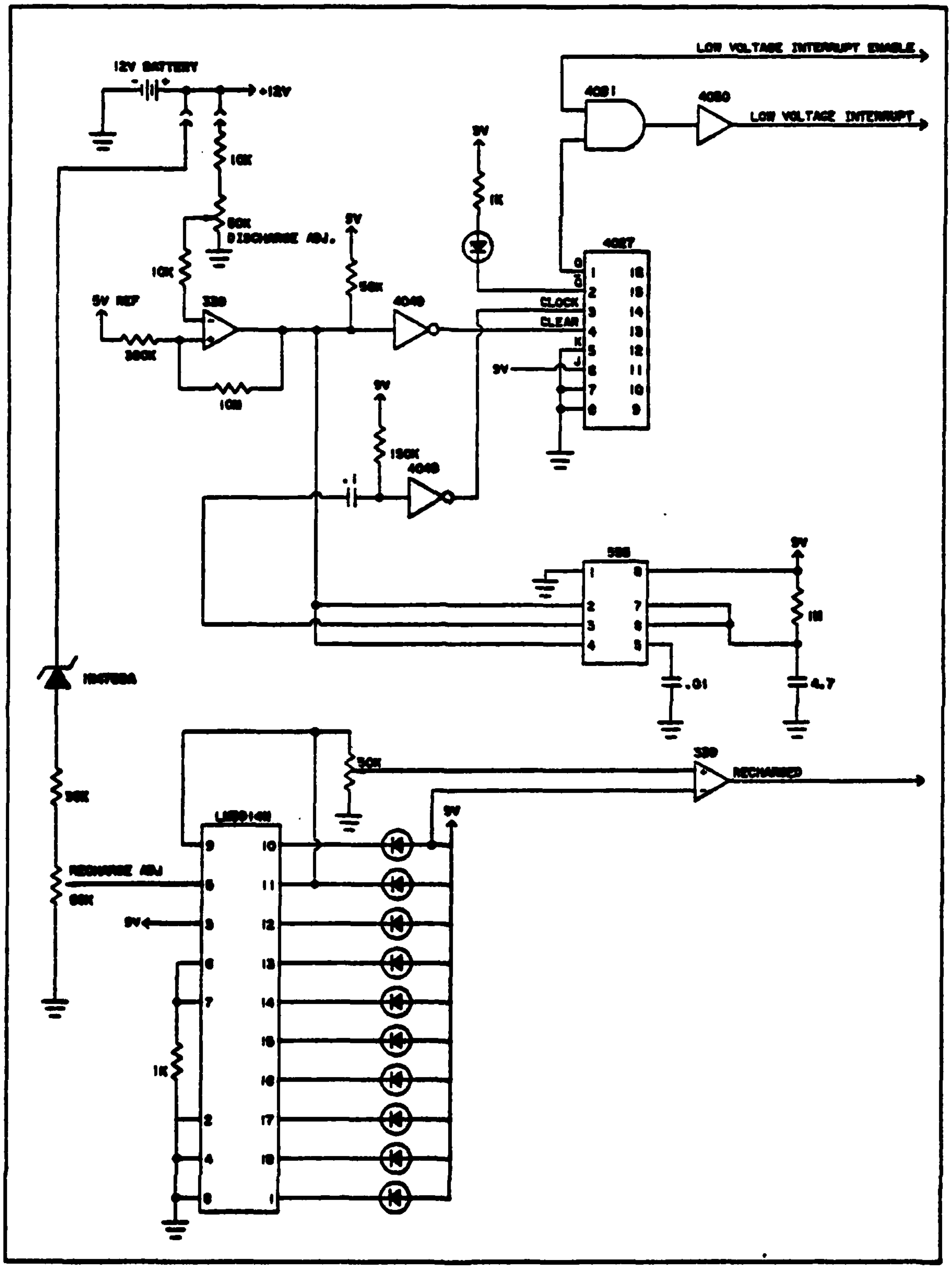


Figure 19. Battery Monitor Circuitry Schematic. Low voltage condition sets flip-flop (4027) after 5 second delay created by 555 timer. LED display gives visual indication of battery voltage level. Fully charged battery is detected by LM339 comparator connected to upper LED. Zener diode establishes display range lower limit of 9 volts (all LEDs off).

D. COLLISION AVOIDANCE

The prototype robot makes use of a six level scheme of proximity and impact detection, implemented through numerous sensors installed at appropriate points on the chassis structure. An active source near-infrared parabolic dish detector mounted on the head provides reliable detection of objects out to a range of five feet, with good bearing resolution (two inches of arc at maximum range). Additional long range information is provided by a forward looking sonar and a ten channel active near-infrared proximity detection system provides close-in protection (out to about eighteen inches). Tactile sensors consisting of projecting feelers at critical points around the base periphery sense impending collisions, and contact bumpers situated all around the base and body trunk alert the CPU to an actual impact. As a final backup, the drive motor current is continuously monitored for an overload condition, indicative of a stalled motor.

The software dealing with all of this sensor data is currently divided into two basic groups, IRQ interrupt routines, and the main program. The main program handles the navigational control of the robot and it is here that the actual planning takes place as required to proceed from place to place in the accomplishment of a desired task or goal. All information available from whatever source is used to this end, and navigational routines are written in loop form to facilitate repetitive polling, sonar activation, and delay timing, with the appropriate exit requirements built into the loop. Loops can be cascaded as needed in the execution of complex navigational routines. Conversely, the software associated with the collision avoidance interrupt routines deals primarily with the sensor data depicting the robot's immediate or close-in environment (such as short range proximity detectors, feeler probes, and impact sensors) and minimal planning is involved. The contents of certain registers can be changed by the interrupt routines, however, to alter the planning processes of the main program after a return from interrupt (RTI). This provides the necessary link for communication between the interrupt routines and the main program and leads toward a more intelligent means of navigation.

The ideal situation would be to have the long range planning executed by the navigational loop be so effective as to make interrupt generation by close-in contacts a rare occurrence, but this would require more numerous long range sensory inputs than currently affordable on this prototype. It is not meant to imply that this method of data analysis for collision avoidance is preferable or recommended for implementation on other systems of greater sophistication: it merely lends itself well to low budget applications involving a minimal number of microprocessors.

The six methods of detection can be broken down into three basic categories: 1) ranging, 2) tactile, and 3) internal.

Category 1 inputs are read by the software making up the navigational loop being executed with the resultant data used to alter course in a planned fashion. Examples are the sonar and the long range near-infrared parabolic dish detector on the head. Also in Category 1 would be the numerous short range infrared proximity detectors, but their inputs are used to generate IRQ interrupts to which the vehicle responds in a preprogrammed reactionary fashion designed to clear the detected obstruction, based on the obstruction location. These responses are implemented within the interrupt routine, upon completion of which control is passed back to the navigational loop, but with the robot hopefully now clear of the obstruction.

Category 2 includes the feelers and contact bumpers, and these also generate interrupts which move the vehicle away from the impacted object.

The drive motor overload sensor falls into Category 3 and serves as a last resort detector, generating an interrupt which reverses the drive wheel and assigns a random steering command to the motor, in hopes of clearing the obstruction. Upon completion of the appropriate pre-programmed interrupt routines, the original drive motor command is restored to the controlling circuitry, and the robot proceeds as before.

The original collision avoidance system consisted of a sonar operating at 21 KHz, built up around National Semiconductor's LM1812 monolithic sonar transceiver chip as shown in Figure 20. Circuit operation is described in

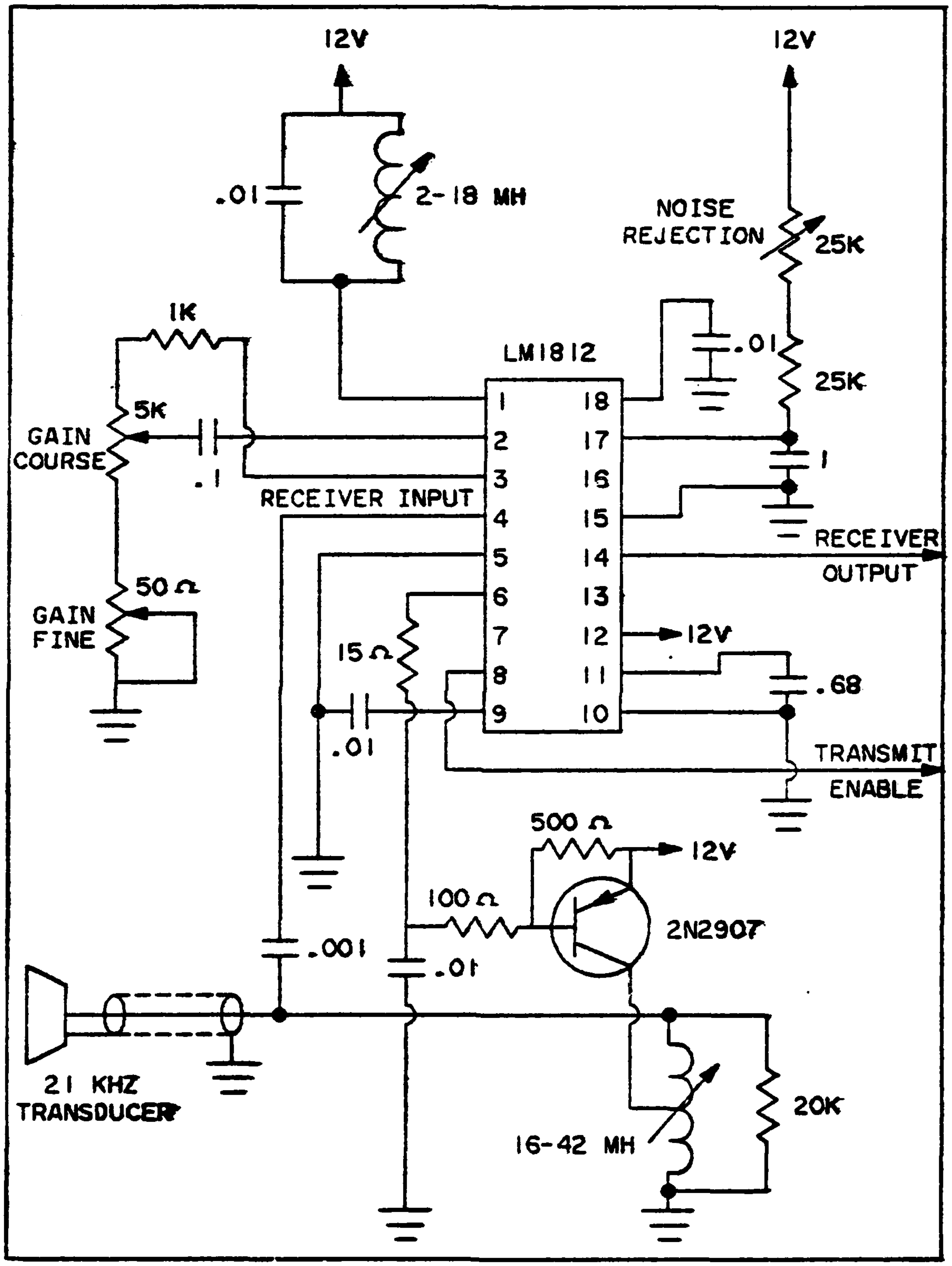


Figure 20. LM1812 Sonar System Schematic

detail by Frederiksen and Howard [Ref. 6]. This sonar provided range information to objects directly in the robot's intended path of travel, and was backed up only by the contact bumper system for impact detection. Initial tests quickly showed the need for greater awareness of the path environment than that provided by sonar data alone, as the LM1812 system was somewhat limited. Although large objects such as walls and furniture were reliably detected, smaller objects often passed unnoticed below the beam pattern. Additionally, no information was provided as to which direction was preferable for a course alteration, since the sonar transducer was mounted to the chassis in a fixed orientation, with no capability to scan back and forth.

The first attempt to gather additional information involved the installation of numerous feeler probes around the perimeter of the robot base, each extending out six to eight inches, and configured so as to provided a normally high TTL compatible output, which went low if deflection was sensed in any direction. These units were constructed from ordinary automobile curb feelers and were flexible so as not to cause damage to either the chassis or objects encountered. The intent was to provide advanced indication of an impact in time to alter course, supplementing the data obtained from the forward looking sonar, and indeed it did provide much improvement, although still crude. Some problems arose, however, from the inertia of the feelers themselves causing false activation of the detection circuitry as they were jarred during vehicle motion and acceleration. The original software was set up in the form of a loop which activated the sonar for transmission, timed the returning echo for object range, and then polled the feelers and bumper switches one at a time for contact, taking evasive action if called for.

A further refinement came with the installation of four near-infrared transmitter/receiver units for proximity detection, oriented so as to provide information on obstacles directly in front of, behind, and to either side of the vehicle, but only within their limited areas of protection. (Each sensor covered a cone shaped region roughly twelve inches out, with a base diameter at maximum range of approximately six inches.) Nevertheless, these sensors proved to be extremely effective in detecting objects within their field of view, and their relative simplicity and low cost made it possible to add six additional units to increase the coverage area. This at the same time increased environmental awareness by better establishing an object's location, as opposed to merely detecting its presence. Nine of the ten units were placed in the first and fourth quadrants relative to the vehicle centerline, for forward protection, as the majority of motion is in the forward direction. Additionally, when collision avoidance routines do call for reverse motion, the vehicle backs into space just previously vacated, and so the odds of a rear impact are greatly reduced.

Improvements to the original detection circuitry ultimately provided an increase in maximum range from twelve inches to thirty inches, and attenuators were added to each detector to allow setting individual ranges and for balancing with other units. Circuit operation is explained in Appendix D. Detectors mounted in a vertical row for the purpose of increasing their field of coverage are simply hardwired to the same input in an OR configuration, while those that provide horizontal resolution of the object location are kept separate from each other and read individually or as vertical groups. These devices performed so well that many of the tactile feelers were no longer needed and were subsequently removed.

The availability of this new sensor information to the CPU made possible more intelligent reactions to impending collisions, but at the same time rendered the polling routine too cumbersome and slow to be practical. At this stage of development Interface Board Number 2 was completed to allow the generation of two channels of interrupts, with up to sixteen inputs each, referred to as IRQ Channel A and IRQ Channel B. Data Selector A which read the inputs associated with Channel A was totally dedicated to collision avoidance devices (see Section I-E). The software was restructured around the interrupt concept, leaving the CPU more time for long range planning.

The forward looking proximity detectors are set for a maximum range of about eighteen inches to give sufficient time for course correction, while the side sensors can see out to a distance of only twelve inches, since less time is needed to react should they detect a return. The design goal was to keep the protection envelopes as small as possible without compromising performance, to allow passage down narrow hallways and between obstructions. Too great a detection range substantially reduces the vehicle's perceived clear space wherein it can navigate, with the result that much time is spent turning circles in the center of the room, unable to exit via a doorway. Even with reduced ranges, situations are sometimes encountered which leave the robot 'boxed in', finding itself trapped between two obstacles and perhaps an adjacent wall. Interrupts generated by infrared returns detected from several sides at once can essentially hinder an orderly exit from this predicament, and what is needed is a means of drawing in the protected envelope until out of this tight spot. There are a number of ways this can be done and the simplest solution was chosen for implementation on the prototype: simply ignore the near-infrared proximity detectors. An even better plan would call for reducing the receiver sensitivity by a factor of two to draw in the envelope as a first step, followed by disabling the receivers altogether as a final resort. This could be easily implemented as a minor hardware change to the threshold detector circuitry.

Less obvious is the problem of how to detect that the situation exists in the first place; i.e., how does the robot know that it is boxed in? The information needed to make this decision can be extracted from data that is already available in memory by appropriate software. The first indication of a problem of this type would be an excessive number of interrupts within a given time frame, say 30 seconds, triggered by infrared returns, feeler and possibly bumper impacts. This is easily calculated by incrementing a register (register irq.fre) each time a collision avoidance related interrupt occurred, and repeatedly clearing this register at specified intervals in real time. (This register clearing is done automatically by the Non-Maskable Interrupt (NMI) routine each time the minutes register is incremented.) Thus, if the register value ever exceeds a specified limit, then the interrupt frequency has exceeded that same limit, since the register is reset to zero every sixty seconds. This register can be checked by the IRQ interrupt service routine each time called.

A second piece of information is needed to verify that a problem does indeed exist, as an excessive interrupt frequency could arise when the robot is navigating a hallway merely from the side returns, without the vehicle actually being boxed in. It was found that tight situations almost invariably were accompanied by rear bumper impacts, which otherwise seldom occurred, as the majority of motion is in the forward direction. So a combination of too many interrupts plus rear bumper contacts is used as the governing criterion and the IRQ service routine responds by disabling the near-infrared proximity detectors. This allows the robot more manuevering room free of interrupts, and thus it frees itself from the trap, relying on tactile sensors alone for guidance. The infrared sensors are later re-enabled by the navigation loop under execution, after a precalculated delay.

An improvement to this scheme would consist of disabling only the interrupt generating capability of the infrared receivers, rather than collectively disabling the receivers themselves. This would allow the sensor data to still be available to assist in a non-interrupt controlled decision as to which direction was best suited for an exit. That this feature would be desired was not clear at the time Interface Board Number 3 was constructed, and the modification was deferred to a later date.

The collision avoidance interrupt software must take into account the overall goal of the navigational routine under execution when taking evasive action to avoid an impact. When docking with the recharging station, for example, this evasive action should not cause the robot to lose sight of the homing beacon. The robot must also be able to tell when a return from a forward looking proximity detector is due to the presence of the recharging station itself, so that the interrupt software does not try to steer the platform away from the battery charger with which it is trying to connect.

The docking routine therefore sets Register Homing to indicate that a docking is in progress, and this register is first checked by the interrupt routines before a response is made. Subroutine Skirt can then be activated by the interrupt routines to effect obstacle avoidance and subsequent realignment with the beacon by the navigational routine itself, rather than by the interrupt routines. When subroutine Skirt is activated, the robot responds by backing away from the charger, and turning until the beacon is positioned 90 degrees right of the forward axis. Then for a predetermined time the platform moves forward, adding hex 7 to the beacon position as seen by the head, and using the result as a steering command for the drive motors. This results in maintaining the beacon at right angles to the direction of travel, in position 0, and the robot moves around the charger to a slightly different position but roughly the same distance away. With the obstacle hopefully now clear, Subroutine Align then realigns the robot with the beacon, and docking continues.

E. INTRUSION DETECTION

The robot's utility begins to develop with the addition of means to detect intruders and unwanted conditions such as fire, smoke and toxic gas. The effectiveness of these detection sensors, their hardware interface, and the corresponding software plays a major role in the determination of the robot's actual worth as an autonomous sentry. The development of increasingly sophisticated but cost effective detection methods which can operate from a moving platform has become a vital area of concern.

On the prototype sentry ROBART, the software maintains the detection and interface systems in one of two modes of operation: Alert Mode or Passive Mode. In the passive mode the majority of sensors are enabled, but a good deal of the interface and drive control circuitry is powered down to conserve the battery charge. The robot mainly relies on passive infrared motion detection, visual motion detection, and hearing to detect an intruder, while at the same time monitoring for vibration (earthquake), fire, smoke, toxic gas, and flooding, etc. Some of these inputs are hardwired to cause an alert (switch from Passive Mode to Active Mode), whereas others must be evaluated first by software, which then may trigger an alert if required. In the Alert Mode, all distribution buses are powered up, and the platform is ready to respond to input conditions, prosecute an intruder, or go on patrol. Either mode can be in effect while recharging and recharging can be temporarily suspended if conditions warrant.

The first intrusion detection system implemented on the prototype involved discriminatory hearing. A bandpass filter is employed to selectively pass along to an audio amplifier those sounds likely to be produced during a forced entry, such as breaking glass, sawing or filing noises, etc. More common household noises, such as those produced by a furnace, air conditioner, or refrigerator, are greatly attenuated by the filter, and therefore do not reach the threshold required to trigger the detector. The detector is hardwired to force the robot from the Passive Mode into the Active Mode if triggered, and this transition is subsequently detected by the software. Directional hearing sensors could also be employed to assist in establishing the location of a detected disturbance, requiring only minor changes to the existing circuitry.

The forward looking sonar used for collision avoidance can also be used in an intrusion detection mode when the platform is not moving by simply recording the range to the nearest obstacle. If an intruder subsequently walks through the sonar field of view and decreases this range, the software can respond accordingly.

The addition of an optical motion detection system to the prototype provided even more capability. Three National Semiconductor type D-1072 integrated circuits are employed to detect changes in ambient light level. These are special purpose chips incorporating a built in photodiode and plastic lens, and are completely passive, requiring no external light source. The cone shaped detection zone created by the lens covers a two foot circle at approximately eight feet, and considerable range is possible, depending on background lighting conditions. The device is capable of operating over the range of 0.1 candlepower to 100 candlepower [Ref. 7].

The sensors are situated with their detection fields oriented so as to cover three slightly overlapping regions, left, right, and straight ahead with respect to the robot head position. This provides a broader detection field, and since the sensor outputs are read independently by Data Selector C, some rough information as to disturbance orientation is available as well.

These optical motion detectors are effective only if the vehicle is stationary, and are automatically disabled when the platform is in motion. Once motion ceases and the software secures the system from an Alert status, the detectors are re-enabled after a short delay to allow them to reset to ambient room lighting. The number of sensor units can be increased to six for 360 degree coverage, as planned for the follow on version to this prototype.

As the software is developed for the overall intrusion detection scheme employed by the robot, it soon becomes apparent that in most environments confirming indication from other sensors should be obtained to minimize false alarms. Since the optical motion detectors must be sensitive enough to respond to light level changes resulting from a person merely walking through the field of view, they are by nature susceptible to false triggering. When incorporated in a fixed alarm system, these sensors may be mounted in locations chosen to prevent exposure to rapidly changing light conditions which might arise from sources other than an intrusion. When mounted on a mobile platform, however, they often may end up pointed directly at a window. In this orientation, it is quite conceivable that the detector could be tripped by the headlights of a passing automobile, or flashes of lightning, for example.

Similarly, overhead flight of propellor aircraft can sometimes intefere with the operation of ultrasonic transducers. Hearing sensors might inadvertently respond to claps of thunder, or sudden noise generated in an apartment overhead by the dropping of an object onto the floor. The robot must be able to distinguish these incidents from a real intrusion situation. If the response to the initial indication is used to direct the attention of all sensors to a possible alarm condition, perhaps through movement of the vehicle to a better vantage point, then all sensors can be polled for secondary indications confirming the likelihood of an intrusion.

A third means of intrusion detection or confirmation is incorporated through use of the parabolic near-infrared sensor described in Section II-D. Mounted on the head, this active sensor was originally intended for locating obstructions around the robot out to a maximum range of six feet for collision avoidance purposes. While this range is rather limited, by positioning itself in the center of an average nine by twelve foot room, the robot can perform a sweep of its head and cover a substantial area. Should a return be present that was not previously there, the possibility of an intruder arises. This sensor is primarily used to confirm a presence previously detected by another sensor, because of its high resolution and maneuverability, and not as a primary detector due to its short range.

The most sophisticated sensor for intrusion detection used on the prototype is a passive true infrared detector sensitive to body heat. This is an off-the-shelf unit manufactured by Colorado Electro-Optics, and has a maximum range of fifty feet. The device is sensitive to any temperature gradient occurring within its field of view such as that created by an intruder walking through the area under surveillance. The detection zone fans out to a width of twenty feet at maximum range.

This unit is intended to be mounted in a stationary position, but was found to be stable enough in an average household temperature environment to operate when the vehicle was in motion, due to the low speed of advance associated with the prototype. A passive infrared sensor, in general, is most sensitive to cross-walk and least sensitive to distant objects moving directly towards or away from the unit [Ref.8].

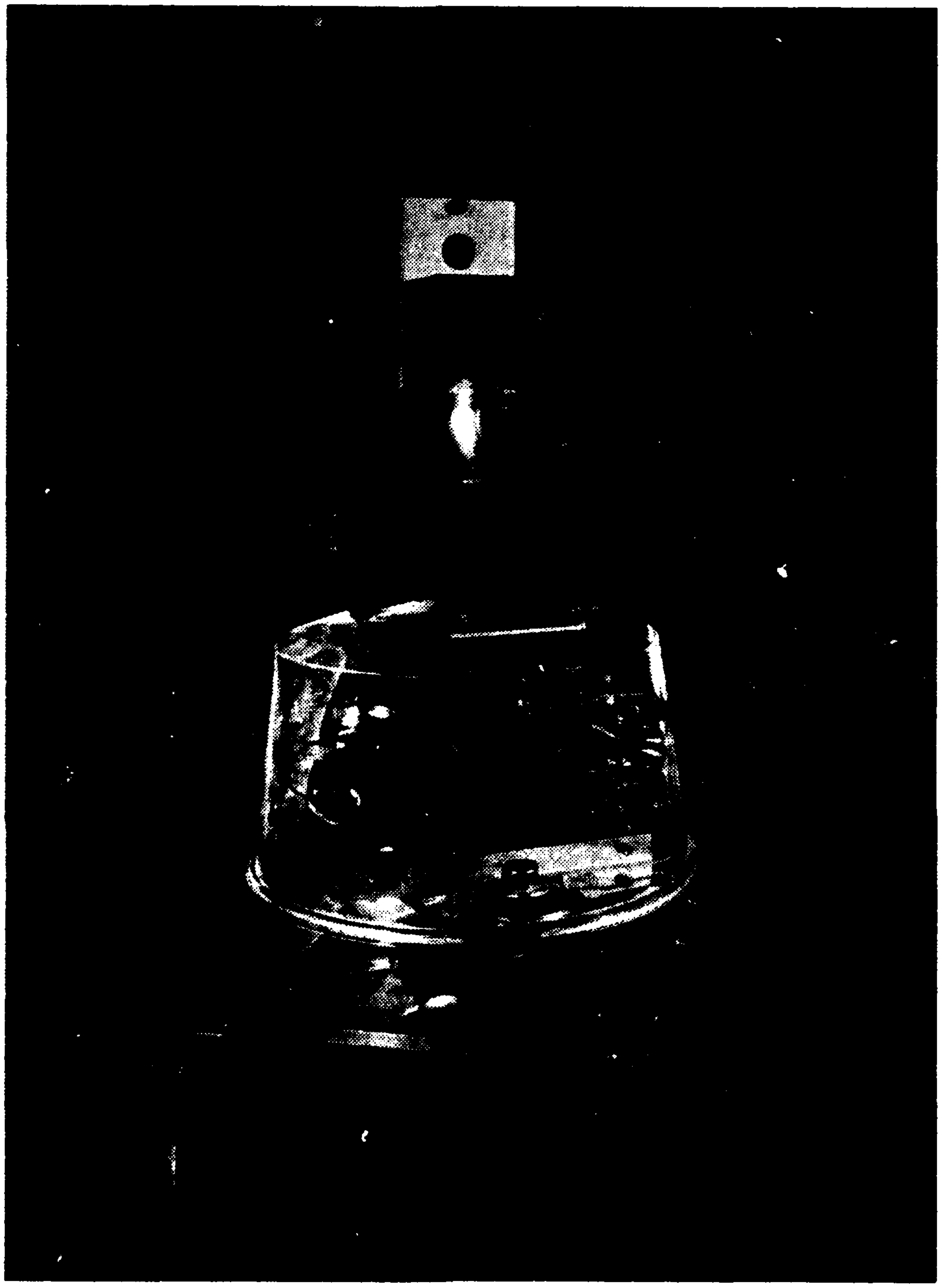
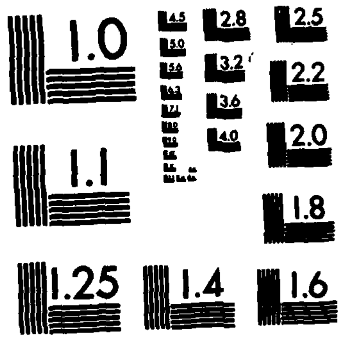


Figure 21. Photo Depicting Sensor Location. Passive infrared detector is visible in white enclosure at top center. Directly below this can be seen the near-infrared parabolic proximity detector. Microphones for discriminatory hearing are located on left and right sides immediately above plexiglass dome. Three optical motion detector chips are mounted directly above the three photocell collimating tubes, inside dome at top center. Speaker is for speech synthesis output.

AD-A125 239 A MICROPROCESSOR CONTROLLED AUTONOMOUS SENTRY ROBOT(U) 2/2
NAVAL POSTGRADUATE SCHOOL MONTEREY CA H R EVERETT
OCT 82

UNCLASSIFIED F/G 17/5 NL

END
FILMED
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

If the head is turned to one side as the vehicle is moving, the presence of an intruder on that side will be detected due to relative motion with respect to the vehicle. An obvious advantage would therefore be realized by mounting a detector on each side of the vehicle, in addition to the one mounted on the head.

In addition to remaining motionless while the platform is moving, the head can be slowly turned while the vehicle is stationary, scanning a circle of fifty foot radius. This would allow the prototype to enter a doorway and stop, and from this position scan the room one time by merely turning its head. An intruder present would be instantly detected, even if able to remain motionless during the sweep.

The use of this infrared sensor in non-stationary applications as described above will vary somewhat with background temperature. Under reasonably cool (65 to 70 degrees F) ambient conditions, the human body, being an excellent emitter of infrared energy, will produce a striking temperature gradient. If vehicle velocity and the rotational speed of the head are kept relatively low, false triggering of the sensor can be minimized. Since the robot reacts to this triggering by merely stopping its motion and bringing other sensors to bear for confirming indications, one or two false alarms create no real problem. After a brief period of waiting the robot will conclude there was nothing there, and resume the patrol.

In the event confirmation of an intrusion occurs, the robot warns the intruder to leave the area through speech synthesis. The software can be structured accordingly to effect whatever action is desired by the operator, but on the prototype version a siren was set off if the intruder was still present after a ten second wait. Exactly what an autonomous sentry should do in this situation depends greatly on the application. A central monitoring station could be alerted via a digitally coded radio transmission that an intruder had been detected, subsequently setting off the building security alarm, and possibly notifying police.

This monitoring station could perhaps keep track of several robots patrolling on different floors, or in different areas of a large industrial plant. Each robot could periodically transmit its location and status, thus providing a means for the central station to be alerted should a robot become disabled. Human guards or other robots could then be dispatched to the scene to evaluate the situation.

In this regard it would probably be more cost effective to provide two robots, one for each floor of a two story building, rather than attempt to create a single robot capable of climbing stairs and intended to patrol both floors. In industrial applications, elevators are likely to be employed within the building anyway, and relatively simple radio links would place these within the robot's control if situations warranted traversing between floors.

F. BEHAVIOR SELECTION AND ALARMS

Once the mechanical functions of a prototype robot have been implemented and interfaced to the microprocessor in such a way that they can be controlled, attention must turn to the software which will dictate the actions of the entire system. Thus far only specific subroutines which manipulate control lines to perform given functions, such as drive wheel positioning, tracking, docking, etc., have been discussed. What is needed now is an overall operating system to call these subroutines in the proper order to accomplish complete tasks, as required by conditions internal or external to the system.

In addressing this need a means should be provided to take into account the priority of the task being performed, and allow termination of that task and substitution of another of higher priority if conditions so dictate. Additionally, in the event no conditions are present which would require action, what then should the robot's behavior consist of? Clearly, to have the prototype continue moving about in a random fashion would be a great waste of battery power, and possibly annoying as well.

In a more sophisticated machine, these problems would be solved as a normal consequence of the software which made up the machine's "artificial intelligence," with a goal oriented program in execution that allowed for recognizing needs, and planning ways in which to satisfy those needs. An elaborate model would be constructed of the machine's perceived environment, as well as its own condition, and through various means the system goals could be broken down into achievable subgoals, with alternate schemes tested first within the model itself, until a workable solution was reached. The train of correct choices which arrived at the desired end point and fulfilled the goal would then be used as a controlling program, and only at this point would it be executed, hopefully meeting with the same success as when the actions were only simulated within the model.

In a system designed around a single microprocessor, however, this is not always possible, unless the number of actions and sensory inputs is kept small enough to allow the program to fit in the available memory space. It was not the purpose of this prototype to serve as a demonstration platform for a very sophisticated AI program, but rather as a test vehicle for different type sensors, their interface circuits, and lower level assembly language routines to serve these sensors. As a natural next step a follow-on robot will be constructed to utilize these already developed concepts, with a more powerful host computer directing the actions of several microprocessors.

Therefore a means was implemented in assembly language software to provide supervisory control of the prototype, to yield a reasonably intelligent process of goal achievement through execution of ordered sequences, each controlled by its own subroutine. Just as the system is equipped with a library of specific subroutines, it is now provided with a set of predetermined behavior routines, each made up of the appropriate subroutines, arranged in the proper order to accomplish a certain task.

The software is structured around one main loop which controls branching to the various behavior routines. These routines each have their own exit requirements, which when met allow return to the main loop, at which point the next routine will be selected. The same routine will not be selected twice in succession, and certain sensory inputs can alter the probability of a routine being chosen. This main loop is first entered via the Cold Start Initialization routine (see Figure 22) when the system is brought on line by the operator.

During this startup voice output is used to announce status as individual systems are energized and tested by the CPU. The operator will be instructed to correct any discrepancies that are detected, such as Enable/Disable switches in the wrong position. Any system failures will cause the CPU to initiate shutdown procedures. Once the startup sequence has been satisfactorily completed, the main loop is entered and a behavior routine selected for execution.

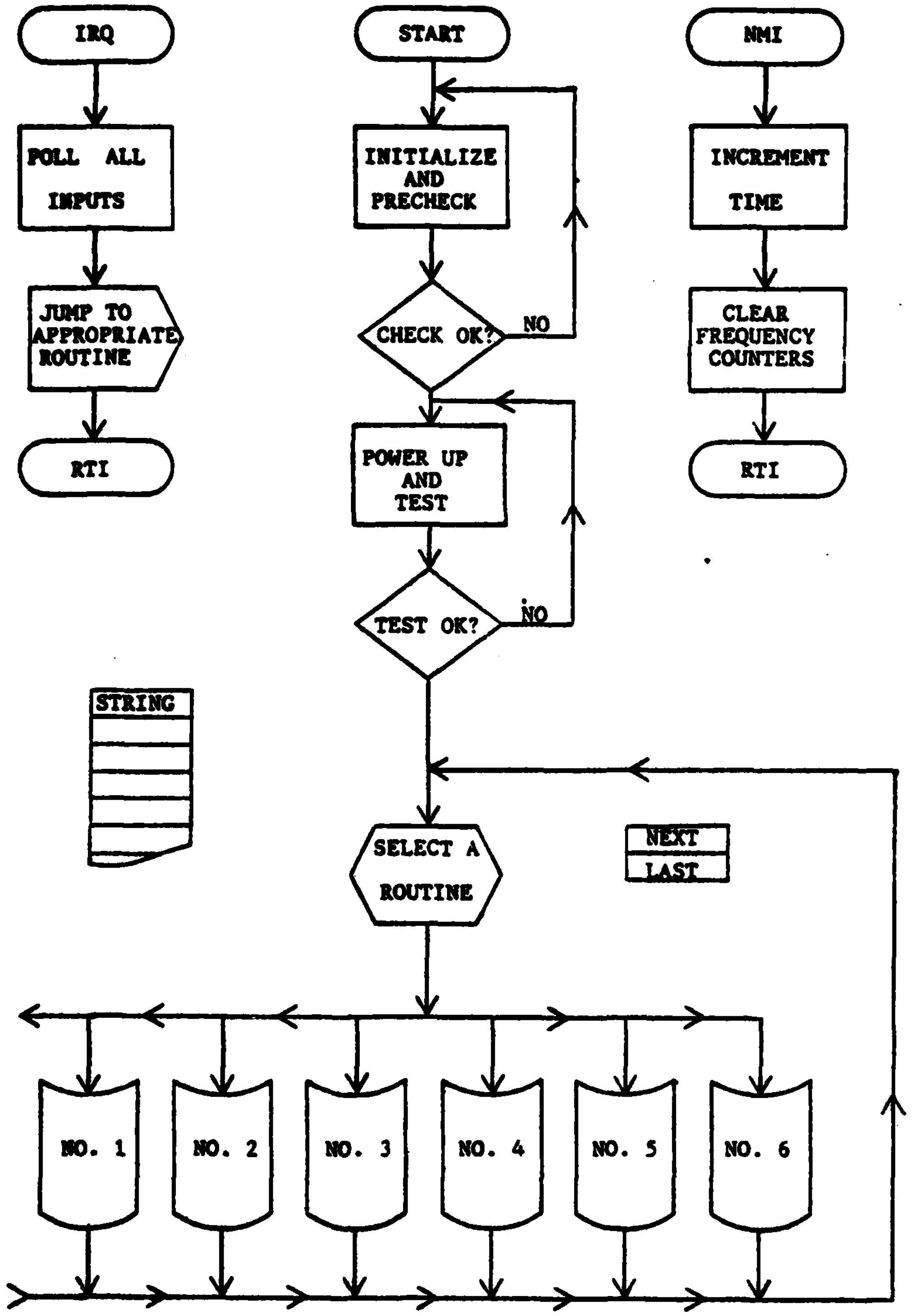


Figure 22. Software Structure for Prototype Robot ROBART. Interrupt Requests (IRQ) deal with all alarms and collision avoidance sensors. The Non-Maskable Interrupt routine is used to implement a real time clock. The main code initializes the system, powers up and checks interface circuitry, and handles behavior routine selection.

While these behavior routines are not in themselves recursive, each does have the ability to call within itself any of the others, which then appear to that routine only as bigger subroutines. In addition, provision is made to call routines in a given sequence. This sequence could possibly have been determined by some goal oriented program which selects a succession of behavior routines, each designed to accomplish an intermediate task required for the achievement of some ultimate goal. The system is always aware of the last routine executed, any routine that was interrupted by one of higher priority, and the next one for execution, if appropriate. If the next routine is not specified, then one is chosen at random, but within the constraints dictated by internal and external conditions. (As an example, non-critical routines involving speech are not chosen if the room is dark, indicating that the household has retired for the night.)

The behavior selection process (performed by the software commencing at label beh.sel) chooses a routine according to an elaborate scheme designed to provide the needed control features previously discussed. Upon initial startup, the first routine is randomly chosen from a possible range of 0 to 15. This range is further decreased to include only routines 0 through 7 if the Drive Power switch is in the off position, as routines 8 through 15 involve vehicle motion.

The number of the chosen routine is stored in Register Last, and the routine is performed until its exit requirements are met. The system is then returned to standard conditions (Passive versus Alert status, with drive system secured, and Interrupt Channel A disabled). If the randomly chosen routine turned out to be unsuitable for performance at the time chosen, as in the case of the example above involving the darkened room, it would not be performed. In any event, the software then loops back for selection of the next routine.

Operator Control Service Number 6 can be used to allow the operator to manually set the contents of Register Next, and thus specify the next routine, within the range 1 to 15. In the selection process (see Figure 23), Register Next is first checked to see if a choice has been specified by this or some other source. If not, then Register String, the first of sixteen consecutive registers, is checked to see if a series of subsequent behavior patterns has been requested. If Register String is not set, then the Hostile/Friendly switch on the prototype front panel is checked. If this switch is in the up position, it will force the selection of Behavior Routine Number 1, which monitors for intrusion (see Section II-E). If none of the above three sources specifies the next routine, however, a random choice is again made, compared with Register Last to ensure no back-to-back duplication, and then performed (see Figure 23).

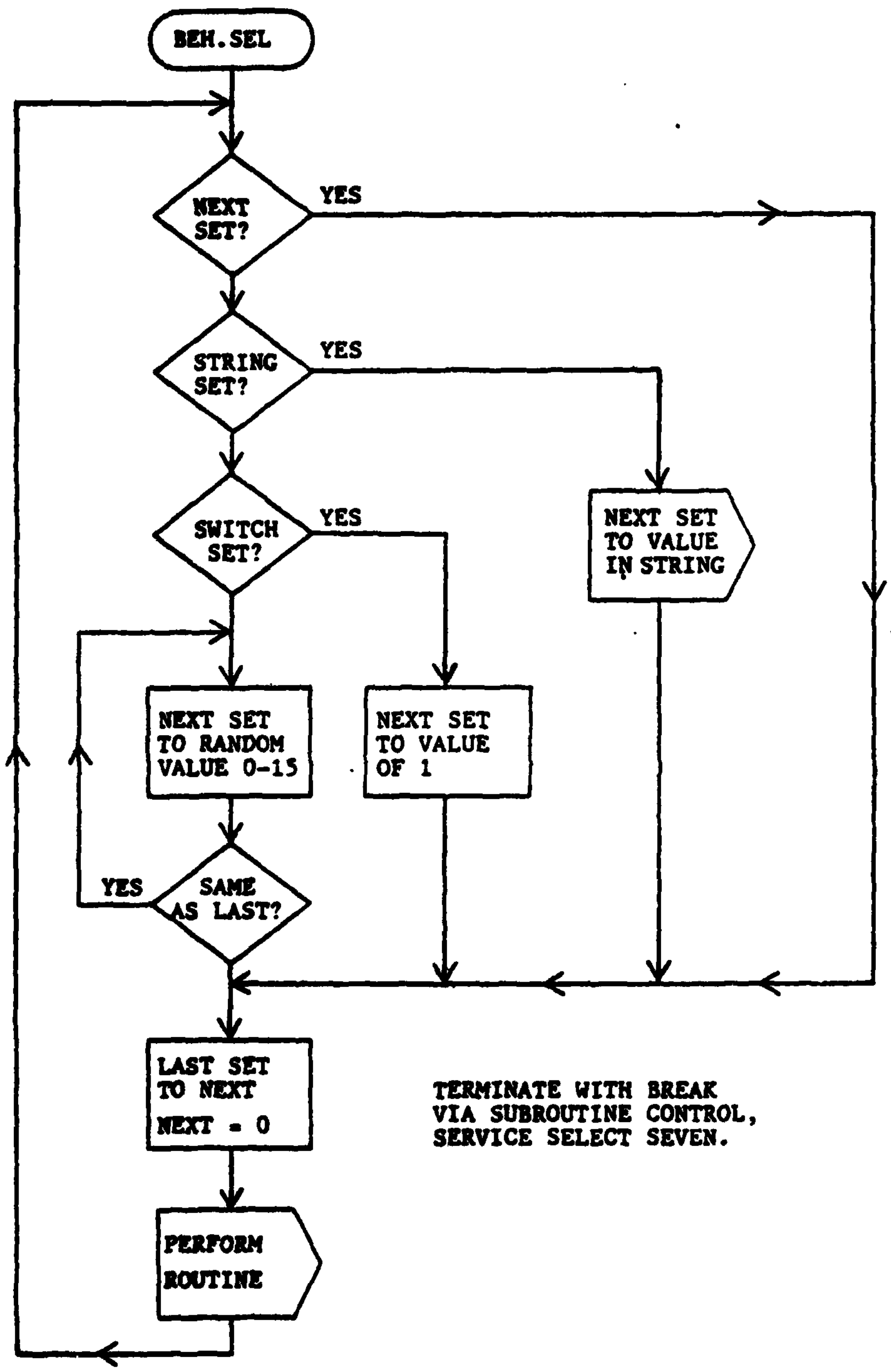


Figure 23. Flowchart Depicting Behavior Selection Process

If Register Next had been set, the routine number contained therein would have been chosen immediately. Alternatively, if Register String had been non-zero, its value would have become the next routine, and each register value shifted down one location in the sixteen consecutive registers beginning with String. (That is, Register String would take on the value of Register String + 1, String + 1 the value of String +2, etc.) The last entry in the list of routines to be performed must be zero, and obviously only fifteen routines can be specified at a time. Each time a value is taken from the Register String, the other register contents each move down one to replace it, until finally String takes on the value of zero ($00) and is no longer set.

The termination of any routine in operation before its normal completion point is a desired feature in this behavior selection process, whether requested by the software itself, or externally requested by an operator. This is implemented through Subroutine Termin, which checks the external ENTRY button and sets Register Exit if the button has been pressed. Register Exit is then checked, and if non-zero, having been set by Subroutine Termin or elsewhere in the software, the current routine is terminated.

When a critical routine is terminated by Subroutine Termin to allow a higher priority routine to take place, provision must be made to allow for resumption of the original routine upon completion of the substituted routine. This is accomplished by saving the interrupted routine number in Register String, and is done by Subroutine Ph.strg. This subroutine manipulates the sixteen consecutive registers beginnings at String so as to provided a stack external to the 6502 in which information can be saved. Just as the register contents move down as routine numbers are taken from String, they move up one register location each time a routine number is saved. This stack is limited in that if more than fifteen routines are saved, the original numbers fall off the end of the stack and are lost. The sixteenth register contents are always maintained at zero to flag the string end.

While the random selection process is limited to routines 0 through 15, the total number of performable routines is limited only by available memory space. The randomly selected routines are designed to produce behavior actions during periods where specific routines, such as recharging, are not needed, and sixteen routines are more than enough for this category. The routines above 15 deal with specific situations requiring definite action on the part of the robot, and can be neither randomly chosen nor selected by Operator Control.

The first of these, Behavior Routine Number 16, deals with all alarm conditions. High priority alarms, such as smoke, fire, flooding, A/D overflow, etc., all cause interrupts via Interrupt Channel B. However, the interrupt routine which polls Channel B takes no action other than to set Register Next to 16, set Register Exit to terminate the current routine in execution, and then disable Interrupt Channel B before returning from the interrupt. Upon return from interrupt, Subroutine Termin in the routine under execution will detect the fact that Register Exit is set, and terminate the routine. Since Next was set to 16, Behavior Routine Number 16 immediately follows, and it decides the appropriate subsequent Behavior Routine to deal with the alarm condition. This choice is based on the value of the interrupt index, Register Q, which was set in the original interrupt polling routine for Channel B. As an example, a low battery condition will cause an interrupt read on input 1 of Data Selector B. With the interrupt index Q set to 1, Behavior Routine 16 will announce the low battery condition through speech synthesis, and then set Next to 17. Behavior Routine 17 then follows, and consists of the beacon acquisition and tracking subroutines, which are used to dock the robot at the recharging station.

It is important to keep in mind that all Channel B Interrupts are disabled by the interrupt polling routine if any of the Channel B interrupt lines go high. Behavior Routine 16, which deals with these interrupts, must either eliminate the interrupt source, or individually disable the input if another routine is going to be called to deal with the source. An example of this latter case is the low battery interrupt. Since the last action taken by Behavior Routine 16 is to re-enable Channel B Interrupts, the low battery interrupt must be gated out first so that it will be ignored while Routine 17 is selected to effect recharging or another interrupt would occur first and recharging could never take place. This is done by hardware circuitry on the Monitor Board. As an alternative example, the smoke detector interrupt is handled by Behavior Routine 16 by simply announcing that the smoke alarm has tripped, and advising personnel to evacuate the room. Behavior Routine 16 then re-enables Interrupt Channel B and returns, and if the detector is still in an alarm state, another interrupt occurs, and the process repeats. If the process repeats more than four times, a siren is activated as a secondary alarm.

Additional routines above 17 can similarly be employed to accomplish specific tasks, just as 17 is responsible for docking with the charger. It should be noted that Routine 17 assumes the robot is in the room with the charger. Another desirable task therefore could consist of going to the room with the charger in it. This theoretically could be preceded by the task of determining where the room with the charger was. These three task numbers could then be loaded into the stack created at Register String for performance in the proper sequence to locate and dock with the charger. If any of these tasks had to be interrupted to deal with a higher priority need, its number would be placed back into String, so that it could be reinitiated when needed, preserving the validity of the sequence. A necessary point to make is that each intermediate task must always be aware of when its particular sub-goal has been reached, and structured so that if a task is scheduled to achieve a result that has already been achieved, the next task will be called. It should be apparent that this arrangement has a great deal of flexibility and potential as a first step in providing realistic supervisory control for a single microprocessor system. The actual Behavior Routines themselves could be stored on a floppy disk, and read in one at a time as needed by the robot as it moved about. This would allow for easy modification and addition of new routines by simply changing the disk.

The stack created at Register String works like a conventional CPU stack from the standpoint that data is always placed on the bottom of the stack, and likewise removed from the bottom of the stack, in a "last in, first out" sequence. However, it would be advantageous in this application to be able to add data to the other end or top of the stack. Routine numbers thus inserted would be performed at the end of the already specified sequence, rather than interrupting it. This is relatively easy to accomplish through a subroutine that merely adds data to the first clear register encountered in the group beginning at String, and in fact the number of registers can be increased if desired to provide for a bigger stack. Considerable sophistication could be achieved with 32 registers.

Now let's assume that there is a separate program in operation, whether on the same or a different computer. This program could be tasked with constantly monitoring the robot's environment and internal conditions, and assigning appropriate goals for achievement accordingly. As previously discussed, it could then work backwards from the desired goal, and establish the required sequence of subgoals, being aware of the Behavior Routines that it has in its inventory and what particular subgoal each was designed to achieve. In a very sophisticated system these Behavior Routines could be theoretically tested first in a system model to ascertain their actual effect. In any event, once the correct sequence of Behavior Routines was determined, their associated Routine Numbers could then simply be placed into the Stack set up at Register String, and they would subsequently be called and performed in the appropriate order to achieve the desired goal. While the Behavior Routines themselves represent canned solutions to specific problems, the sequence in which they are performed is determined by the robot based on the overall objective. It was the purpose of the first phase of this prototype development to provide the means to implement the chosen sequence, while the supervisory program which determines the sequence is to be developed at a later date.

G. NAVIGATIONAL PLANNING

The ten near-infrared proximity detectors described in Section II-D are sufficient in themselves to allow the vehicle to make random patrols of a household. The navigational loop simply instructs the prototype to move directly forward as long as no obstructions are detected, and the collision avoidance system takes over to avoid obstacles as they come into view of the sensors. The resulting motions of the vehicle will eventually carry it from room to room, but in a totally unpredictable and extremely inefficient fashion.

Obviously a robot intended to serve in a security role must be provided with an intelligent means of navigating, if for no other reason than to minimize drive and steering motor power consumption. It would be no great challenge to outsmart a mechanical sentry that relied on random motion to carry it from place to place. Therefore the development of a more sophisticated means of determining the vehicle's course becomes an important milestone in its evolution toward true autonomy.

The most effective approach to solving this problem involves the development of a memory map wherein the machine can encode information about its environment as it moves about. A simple way to do this would be to assign a single byte in memory to each square foot of floor space, as presented by Weinstein in the book "Android Design" [Ref. 9].

An average room ten feet by twelve feet would require only 120 bytes, and thus an array representing an entire household floorplan could reside in less than 2 kilobytes of memory space. Over a period of time the robot could fill in the originally blank map with probability codes reflecting the chances of finding an object in a particular square. For example, a code of zero could indicate that there has never been an object detected in that location. Code 1 might mean there was once, but not always, code 2 indicating there probably is, and code 3 meaning there always is an object in that square. The robot can reassign probability codes as conditions change over a period of time. A special code could be preassigned by the system programmer to indicate areas the robot must never traverse, if so desired.

With a memory map of this type algorithms could be developed similar to those used in computer games (such as Othello) to locate clear pathways through a room full of obstructions. The geographical position of the recharging station could be recorded, as well as door openings and hallways. Planned routes for patrolling could be preprogrammed, or generated by the software.

The main hurdle to the implementation of a scheme of this sort results from the fact that this concept requires the machine to know its location at all times, as well as its orientation in that spot. There presently exists no inexpensive way to provide this capability. Various means have been suggested, such as using sonar to establish the minimum range (and hence a perpendicular line) to each wall, and through geometry calculate position in terms of memory map coordinates. While this sounds plausible in theory, a glance around the average room reveals a significant number of drawbacks in the form of doorways, windows, and upright furniture which would invalidate the sonar data and greatly complicate the needed software.

Therefore, as a first step the prototype was given the ability to create a relative rather than absolute model of the objects around it. By turning its head and recording the position of obstructions within the field of view of sensors mounted on the head, as it rotates from side to side, the robot can then examine this much less sophisticated memory map for information previously unavailable. This allows the vehicle to navigate in a far more effective manner while efforts continue to develop hardware to assist in absolute position referencing.

This relative model requires only 16 bytes of memory space, one for each of the points of resolution of the head position A/D converter. Since each byte consists of eight bits, eight pieces of information can be stored for each pie shaped sector of the robot's perceived world (Figure 24).

The most obvious piece of information needed would be the presence of a near-infrared return as detected by the

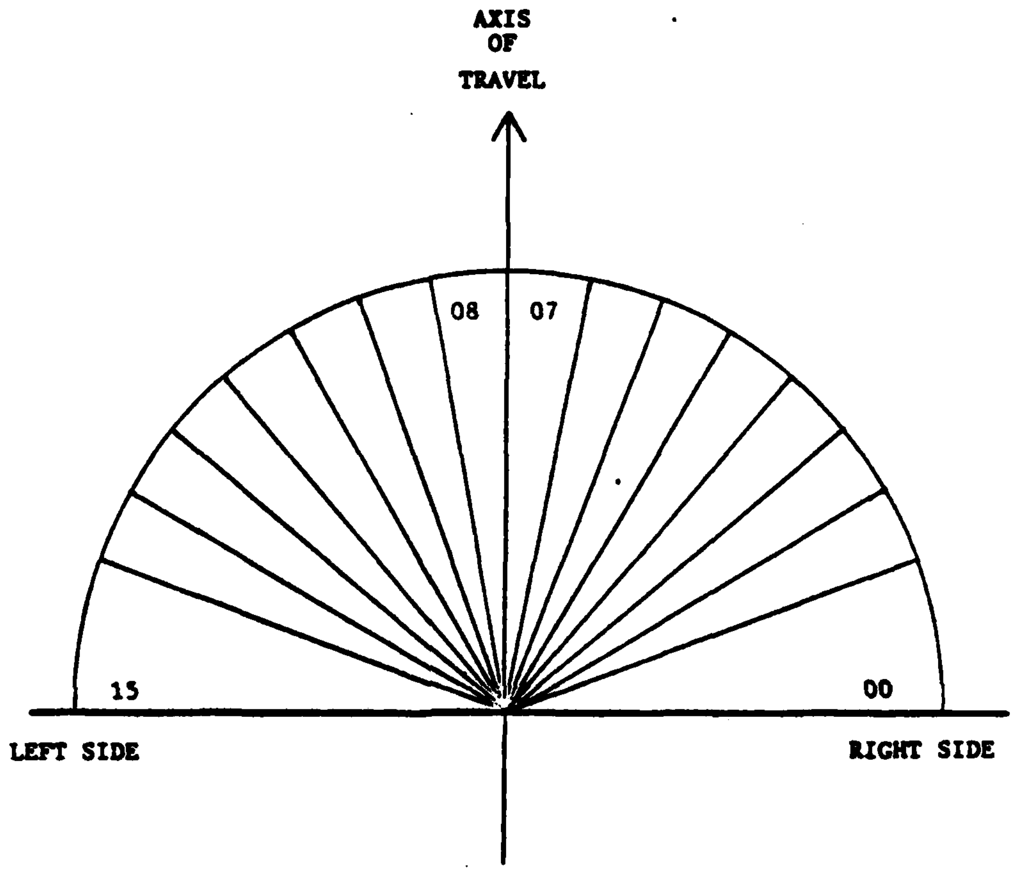


Figure 24. Relative Model of Robot's Perceived Environment. A specific location in memory is assigned to each of the sixteen sectors within the field of view of head mounted sensors.

parabolic dish sensor, mentioned in Section II-D, which has a range of six feet. Any object out to that distance would be recorded by setting the designated bit in the memory location assigned to that specific head position, say bit 0. Bit 1 could represent the presence of a bright light source when set, bit 2 could be set if the infrared heat detector output went high, bit 3 could represent a sonar return from a transducer mounted on the head, and so on.

To avoid confusing the issue, in the following discussion only two pieces of information will be considered: a near-infrared return from the parabolic sensor, represented by setting the lower four bits, and the presence of a bright light source, represented by setting the upper four bits of the appropriate memory location.

The first step in the implementation of navigational planning based on this relative model involves writing a subroutine (Subroutine Survey) to turn the head to position zero (full right) or position fifteen (full left), whichever is closer. The head is then swept once through the full range of positions, and the software polls the appropriate sensors in a repeating loop until the sweep is complete (3.5 seconds). If any sensor output is found to be high, the head position is read, and the appropriate bits set in the corresponding memory location. Upon completion of the sweep the data in memory can be used in decision making subroutines that ultimately dictate the robot's actions.

One such subroutine is Subroutine Sort, which runs through the memory locations and determines the starting and ending boundaries of any obstacle-free zones, expressed in terms of head position. This can be followed by Subroutine Choose, which selects the largest free zone, and Subroutine Center, which calculates its midpoint. This resulting value can be used as a steering command, maneuvering the vehicle into uncluttered space away from obstructions. It is also very useful in locating doorways.

Figure 25 shows a sample printout of a test harness written during the development of these subroutines. The register contents following a sweep of the head under control of Subroutine Survey are listed, and immediately below are shown the starting and ending boundaries of the two largest obstacle-free sectors. The larger of these sectors is then chosen and its midpoint subsequently calculated, as marked on the figure.

While the time required to update the contents of this relative memory map is only 3.5 seconds, it is still long enough to allow the robot's position and orientation to change considerably. Changes in position have minimal effect on the validity of data obtained while the vehicle is in motion due to its very low speed of advance (.26 feet per second). Changes in orientation can be significant, however, due to the sharp turning angles possible (up to

```
OUTPUT DATA FROM TEST ROUTINE                 29 JULY 82

Test of relative model data aquisition
Model contained in registers $30 - $3f

run test                >run test
00 00                   00 0F ┐
01 00                   01 0F │
02 00                   02 0F │
03 00                   03 0F ├──────── Sector Obstructed
04 00                   04 0F │
05 00                   05 0F │
06 00                   06 0F ┘
07 00                   07 00
08 00                   08 00
09 00                   09 00
0A 00                   0A 00 ┐
0B 00                   0B 0F │
0C 00                   0C 0F ├──────── Sector Obstructed
0D 00                   0D FF │
0E 00                   0E FF ┘
0F 00                   0F F0

00 0F    00 00          07 0B     00 00
                                          Center of Largest
00 0F    07             07 0B     09  ────Free Zone
                        ─────────────────Largest Free Zone
>run test               >run test
00 00                   00 00
01 00                   01 00 ┐
02 00                   02 0F │
03 00                   03 0F │
04 00                   04 0F │
05 00                   05 0F ├──────── Sector Obstructed
06 00                   06 0F │
07 00                   07 0F │
08 00                   08 0F ┘
09 00                   09 00
0A 00                   0A 00
0B 00                   0B 00 ┐
0C 00                   0C F0 │
0D F0                   0D F0 ├──────── Bright Light Source
0E F0                   0E F0 │
0F F0                   0F F0 ┘

00 0F    00 00          00 02     09 0F

00 0F    07             09 0F     0C ──── Center of Free
                                          Zone (09 - 0F)
```

Figure 25. Sample Output of Test Harness

80 degrees). The problem is further complicated as the sweep could be in the direction of the turn, or opposite to the turning direction. The result is invalid data in either case.

An additional complication can arise if an interrupt should occur during the sweep. The head is turning under hardware control, and will continue to do so even if the CPU is called away to service an interrupt. Interrupts cannot be disabled during the sweep because the close-in collision avoidance strategy is built up around the near-infrared proximity detectors which all generate interrupts. Therefore there exists the very likely prospect that regions free of obstacles could be recorded which did not really exist, simply because the CPU was busy with an interrupt and did not get around to polling the sensor. Since only 250 milliseconds are available during the sweep for processing information on each pie shaped sector of the model, as compared with the five or six seconds needed to execute some collision avoidance interrupt sequences, the problem is significant.

The first problem arises due to the length of time involved if the head must sweep out the entire domain of sixteen sectors. A more practical approach would be to limit the sweep to those sectors directly in the intended path. Since the sweep limits are determined by software, this is easily implemented. Subroutine Radar sweeps the head back and forth between sectors 6 and 9, and the robot moves forward as long as no objects are detected. If an object comes into view on the left, the course is adjusted slightly to the right, and vice versa. Should the entire four sector region become blocked, then the vehicle comes to a halt and performs a complete sweep in an attempt to locate a clear path.

Since the total time required to update this abbreviated model is significantly less than 3.5 seconds, and since turning angles are kept within 40 degrees as opposed to 80, the chances of loading invalid data are greatly reduced. There remains only the task of integrating this long range navigational planning process with the close range collision avoidance system.

Essentially what is needed is a means for the navigational loop to know when an interrupt has occurred, so that the data in the model can be ignored, and the model updated. Since this requirement was anticipated at the time the interrupt routines were written, the solution is relatively simple. Register IRQ.num keeps track of the number of collision avoidance related interrupts, incremented each time by the IRQ routine itself. Therefore, if this register is checked at the beginning of an abbreviated sweep, and its value is the same upon completion of the sweep, then the data recorded during the sweep is valid, as no interrupt occurred to distract the CPU. If the value is not the same then the model is cleared, and reflects no obstructions present.

This may indeed not be the case, but the situation will be remedied quickly as the next sweep is performed, provided another interrupt does not take place to invalidate it also. The robot moves straight ahead during the interim. If another interrupt does occur, however, the collision avoidance routine takes over anyway, and so the robot responds under interrupt control to move away from the obstruction. Once clear, the software returns to the navigational loop and the sweep resumes.

Subroutine Radar can be called while the vehicle is in motion, for advanced information on what's out ahead. Interrupts are not disabled during the sweep, but rather given priority, with the sweep information ignored if an interrupt occurs. Subroutine Survey, on the other hand, is called only when the platform is stationary, for a complete picture of the surroundings, and interrupts are disabled for the 3.5 seconds required for the full sweep. This approach allows the two systems to work together without conflict, the longer range infrared sensor yielding to the close range proximity detectors when a collision threatens.

The robot now has the ability to look out and see obstacles four to five feet in front of it, and subsequently alter course so as to pass to one side. Should it become boxed in, it has the ability to stop and scan through the entire range of head positions for a clear zone. This offers a great improvement over the purely random motion of before, but still leaves a lot desired, if patrols are to be made in an orderly fashion.

Again, what is needed is a means of determining absolute position and orientation. Until such time as this can be practically implemented, the robot must make do with the information it has available. For a start, it can always reference off the position of its recharging station, which it can locate and positively identify. Secondly, hallways, being long and narrow, are relatively easy to recognize. If the household floorplan allows for positioning the recharging station where it can be used to advantage in locating the hallway entrance then the robot can systematically find the hallway and proceed down it, stopping at each doorway to check adjoining rooms while on patrol. The robot can determine its orientation in the hallway if told beforehand which direction affords a view of the beacon on the recharging station. With prior knowledge of where the rooms are with respect to the hallway, the robot can proceed accordingly.

III. HIGH LEVEL LANGUAGE FOR ROBOT CONTROL

A mobile robot or platform is inherently more flexible than a physically restrained intelligent machine of the kind typically found in industrial applications. This is not to imply mobility is more desirable, and in fact in many applications it would unduly complicate things rather than improve them. For this reason, almost all research to date has been addressed towards fixed location devices and the control of their attached manipulators and end effectors. Here industry provides an immediate market and subsequently motivation for research aimed at improvements in design and application. In response to this stimulus high level languages for robot control have emerged. These, although still primitive and not altogether general, have indeed simplified control and programming in the systems for which they were written.

While there exists a multitude of applications for mobile robots, the very nature of mobility dictates the use of greatly expanded sensory input, particularly in the case of a fully autonomous machine operating in unknown or even changing environments. Our present technology offers no cost effective means of implementing these sensory systems except where operation in hazardous environments intolerable to humans can be used to justify the extremely high costs.

As a result, fewer systems are in use and development lags those areas offering immediate monetary return.

It soon becomes obvious that the inherent flexibility of a mobile system should not be compromised by the lack of a high level language to support it. Operations and functions performed by the system can best be implemented, terminated, or altered by either an operator, a programmer, or the system itself when the specifics involved are not addressed. For example, it should only be of concern that a left turn is called for, and not which values need be assigned to what registers, I/O ports, and data distributors to effect that turn. Nor should the actuator control system that maintains the steering wheel in the position called for come into play, whether it be implemented in hardware or software. All these details should be buried in the lower levels of the hierarchy, invisible at the top where decisions are made.

For reasons mentioned earlier, a universal high level language strictly for mobile robot control does not exist. To support the work reported here, a high level language was developed for control of the demonstration platform, the prototype robot nicknamed ROBART. Due to time and monetary restraints, and anticipated generalizations of the language, no attempt has yet been made to write a compiler. Instead, its use is simulated through subroutines acting on previously loaded registers in such a way that only minor changes need be made to make use of a compiler or interpreter should one be developed.

It should be noted, however, that compiling a language for robot control is somewhat more involved with system specifics than is immediately apparent. Compilers for conventional computer languages can be somewhat generalized by structuring in such a way that only the input/output addresses need be changed to adapt from one type of computer to another. As a simplified example, the command PRINT is universally used to take information from a buffer and output it to an external device. Only the buffer location and the output port address will change from one system to another, and methods exist to facilitate handling that change. In a robotics application, the computer to hardware interface must deal with much more than just a printer, keyboard, CRT, and mass memory, and so there are many more possibilities for change. One arm alone, for instance, can have six or more actuators, not to mention the sensory inputs needed for its operation. Additionally, the hardware now addressed is much less standardized than conventional computer system hardware. Most printers, for instance, have either a parallel interface or a serial interface for data transfer, and are often directly interchangeable. Such is not the case with robotics hardware. The use of stepper motors to position an arm requires an entirely different scheme of control than does an arm designed around DC servomotors, and therefore the command RAISE ARM must result in completely unrelated sections of code for the two systems. The question must also be addressed as to whether the controlling computer is talking directly to the arm positioning motors, or perhaps to a dedicated microprocessor which in turn then talks to the motors, i.e., an 'intelligent' arm. The net result may very well be that while the language itself could be standardized, the invididual compilers would have to be unique for their own particular application. A way around this would be to have the language address only the controlling computer, and standardize the controlling computer's interface to its dedicated microprocessors for input and output. This would not be cost effective for small systems, and is not likely to happen anyway.

This robot control language in its preliminary stage of development consists of operators designed to perform a particular function, usually discernible from the operator name. For example, the operator STOP terminates a process previously begun. Exactly which process depends on the parameter or group of parameters (not to exceed three) following STOP. These parameters can be variables or constants, and are not needed for all operators. A list of operators and their associated parameters, if any, is given in Appendix E.

Operators can be listed sequentially as statements in a program which in turn controls the motions of the robot. This program should not be confused with the high level artificial intelligence program which ultimately decides the behavior of the robot, and for which languages already exist, such as LISP or ADA.

V. CONCLUSIONS

The prototype robot ROBART, begun in August 1980 and completed in September 1982, was intended to function as a development platform for an autonomous robot sentry, with emphasis on testing appropriate sensors and their associated interface circuitry and software. The physical structure of the prototype therefore was chosen to allow easy access to internal components and circuitry, and consequently is not suitable for extended unsupervised operation in a normal home environment. A body design hardened for survivability and free of external projections likely to snare on or be damaged by nearby objects is needed before a follow-on version of this prototype can be practically employed.

The behavior selection process discussed in Section II-F provides the means for execution of the end results of some goal oriented artificial intelligence program which in fact could be running on a separate computer. This higher level program could be tasked with evaluating the robot's environment and needs, establishing goals, and then creating a software model in which to test the primitives represented in the various behavior routines. Each of these routines would be designed to allow achievement of a specific subgoal under given entry conditions. When an appropriate sequence is found which reaches the desired goal, the corresponding routine numbers are pushed onto the stack and then performed in sequence. Since no higher level artificial intelligence program for determining sequences has yet been implemented on the prototype, the routine numbers are specified by interrupt service routines, based on the nature of the interrupt.

V. RECOMMENDATIONS

The development of this first generation model resulted in a fairly sophisticated single-microprocessor robot. However, results clearly indicate the need for future versions to employ additional microprocessors to allow the controlling CPU more time to devote to overall coordination and planning. The specific functions of head positioning as well as steering and drive control currently implemented through rather limited logic circuitry could be better performed by individual dedicated microprocessors. This would allow a full eight-bit resolution analog to digital conversion for the head position, resulting in 256 discrete position increments, more than enough accuracy for processing information from head-mounted sensors. In addition, precise position error feedback would make velocity control of the head possible during positioning.

With a microprocessor dedicated to head positioning, the circuitry contained on the Optical Board and Interface Board Number 5 could be upgraded to make use of eight-bit analog to digital converters to monitor each photocell output from the optical array, as well as the ambient light photocell output. The eight channel National ADC0808 A/D converter would be ideal for this application, with four channels left over for other uses. A tracking system comparing digitized photocell output to keep the head centered on the beacon would yield accuracy far greater than that achievable with the current circuitry.

For the second generation prototype a steering and drive system utilizing two independently driven center wheels will be employed, with caster-type idlers at front and rear. This will eliminate the need for a separate steering motor and its associated position sensing A/D converter. Motor direction as well as individual motor speed can be precisely controlled through pulse width modulation circuitry, allowing an almost infinite range of turning radii for use in navigational and docking routines. A separate microprocessor specifically tasked with controlling the pulse width modulation and monitoring the actual motor speed could provide precise turn radius, advance and transfer information to other dedicated microprocessors as well as to the controlling computer.

An additional microprocessor assigned the task of determining vehicle location could also maintain an updated memory map of the surroundings, noting the locations of the battery recharging station, doorways, and other relevant items, in addition to obstructions. Dead reckoning information could be obtained from the drive and steering controller and combined with actual position information taken in by sensors and appropriate hardware. These memory maps could be down-loaded onto an onboard 5 1/4" disk when the robot left the area represented. As it moved into a new space a previously stored map of the new area could be substituted. In this manner an entire floorplan could be represented by a set of easily manipulated rectangular map arrays. The development of cost effective hardware to precisely determine position and orientation remains a crucial first step towards this end.

The same disk storage device could also be employed to store all or at least some of the behavior routines or primitives. This would make considerably more routines available for the robot's use, and at the same time separate the programming requirements for the routines themselves from that of the software which implements the routines. Behavior routines could be added or deleted by changing the software on the disk only, with no modifications to the robot operating system software which loads and executes the routines. The routine numbers specified by either interrupt routines or a high level artificial intelligence program would correspond to the file numbers of the routine software as loaded on the disk. The sequence of files to be loaded and executed would be stored in the stack created at Register String as discussed.

For maximum efficiency the collision avoidance strategy in its entirety could be delegated to the supervision of a separate processor. A multitude of information from tactile and proximity sensors could be preprocessed before being passed on to the controlling computer for integration with the navigational planning software. In the prototype it was found that excellent protection could be obtained with the majority of near-infrared detectors oriented for forward coverage. Sensors situated at 6, 18, and 28 inches from the floor in a vertical column proved very adequate for obstacle detection. Columns situated as depicted in Figure 26 would give excellent coverage of the platform's surroundings with sufficient resolution of a detected object's location for appropriate evasive reaction.

It was found that these near-infrared proximity detectors are better suited to this collision avoidance application than ultrasonic sonar units. The divergence of an acoustical beam by far exceeds the narrow cone of radiation from the high powered LEDs, and focusing or collimating devices can be employed to further increase the resolution of the near-infrared detectors. Additionally, false triggering of these devices seldom occurs, whereas more sophisticated circuitry is required to eliminate reactions to spurious signals with conventional acoustical transducers. The directional characteristics of these infrared sensors make it possible to operate numerous devices in proximity to each other without cross coupling, which is again much harder to eliminate with multiple sonar units. The simplicity and low cost of the near-infrared sensors makes

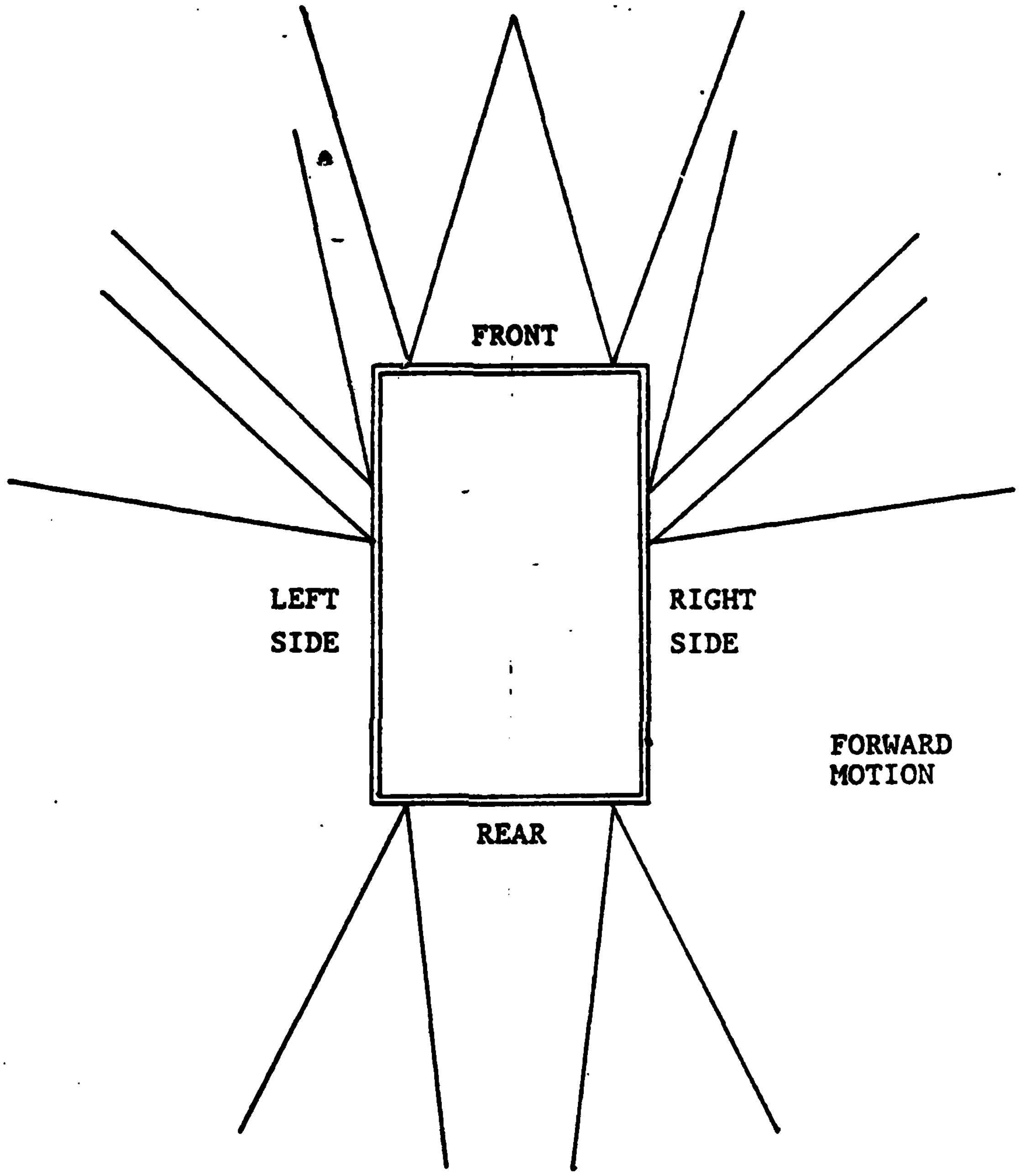


Figure 26. Infrared Proximity Detector Sensor Arrangement. Top view of robot depicting cone-shaped areas of coverage for vertical columns of proximity detectors, with majority of sensors oriented for forward protection.

possible peripheral coverage to whatever extent desired for environmental awareness with regard to surrounding obstructions.

A proposed block diagram for this multi-microprocessor system is shown in Figure 27. Assembly language programming is well suited to the needs of the individual dedicated controllers, yielding fast and efficient code for the direct manipulation of output control lines and/or the reading of sensory inputs. The controlling microprocessor could also be programmed in assembly language, or even a robot control language as discussed in Section IV. This intermediate microprocessor would serve as an interface between the robot's subsystems and an eventual goal oriented artificial intelligence program written in ADA or LISP.

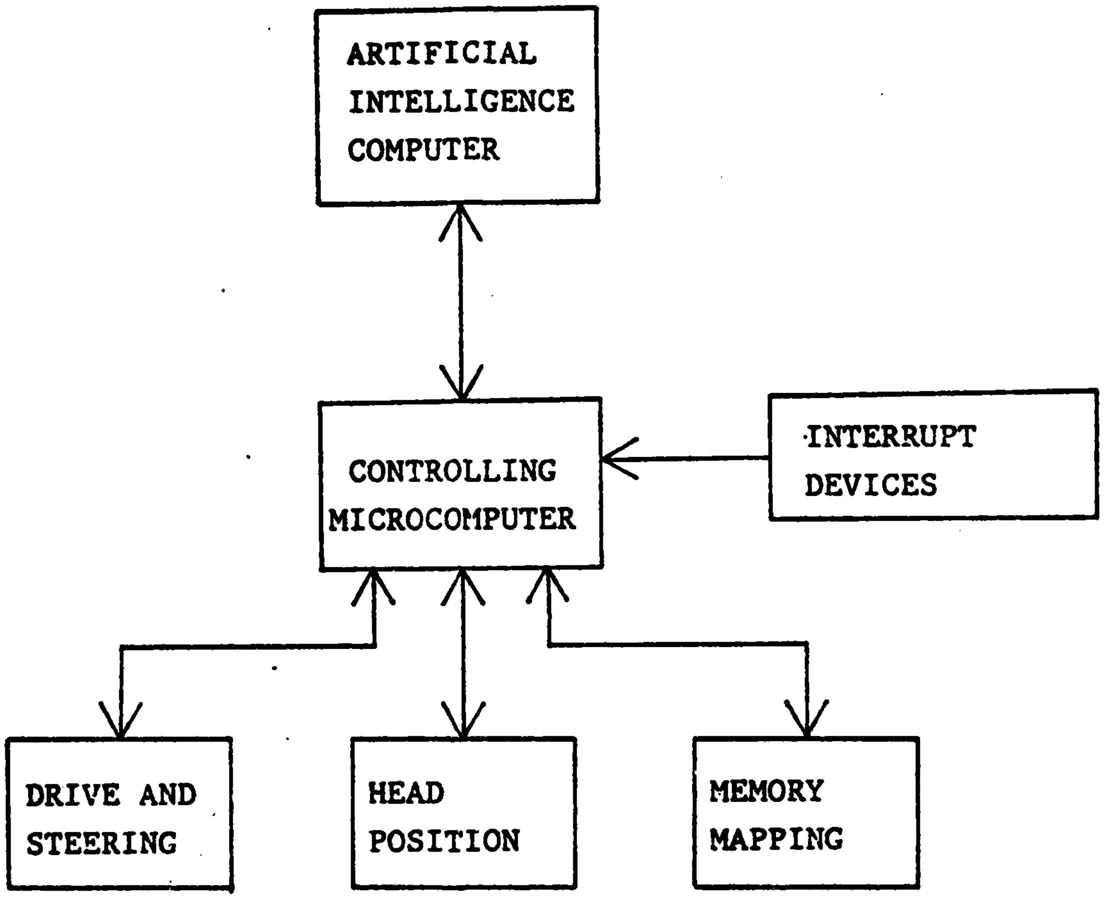


Figure 27. Proposed Layout of Multi-microprocessor System. Individual dedicated microprocessors would be assigned specific tasks such as drive and steering control, head position control, memory mapping, etc.

APPENDIX A

INPUT/OUTPUT PORT ASSIGNMENTS

6522-2 Versatile Interface Adaptor

ora2	AA Conn	Function
PA0	0D	B0 steering command
PA1	03	B1 steering command
PA2	0C	B2 steering command
PA3	12	B3 steering command
PA4	0N	Drive Power Relay (1 = on)
PA5	11	Drive Direction Relay (1 = forward)
PA6	0M	Data read B (yellow)
PA7	10	Data read A (red)

orb2	AA Conn	Function
PB0	0L	C0 selector address (yellow)
PB1	09	C1 selector address (red)
PB2	0K	C2 selector address (green)
PB3	0B	C3 selector address (white)
PB4	0J	Data Write 1
PB5	07	Data Write 2 (white)
PB6	0H	Data Write 3 (green)
PB7	06	Head position latch enable

6522-1 Versatile Interface Adaptor

oral	A Conn	Function
PA0	14	Sonar B0 address
PA1	04	Sonar B1 address
PA2	03	LED for sonar echo
PA3	02	unused
PA4	05	CPU power switch verify
PA5	06	RS-232 connection verify
PA6	07	Printer busy signal
PA7	08	Sonar receiver (goes low on echo)

orb1	A Conn	Function
PB0	09	unused
PB1	10	unused
PB2	11	Speech synthesis Chip Select (CS active low)
PB3	12	Speech synthesis trigger (R/W)
PB4	13	A1 speech synthesis address
PB5	16	A2 speech synthesis address
PB6		not available
PB7	15	Speech Synthesis Busy (1 = busy)

6522-3 Versatile Interface Adaptor

ora3	AA Conn	Function
PA0	0V	Write protect Monitor RAM
PA1	0W	D0 head position (white)
PA2	0X	D1 head position (green)
PA3	18	D2 head position (white)
PA4	19	D3 head position (red)
PA5	20	Interface power up (1 = off)
PA6	17	Data read C
PA7	0U	ENTER input (red)

orb3	AA Conn	Function
PB0	16	B0 switch 1 (yellow)
PB1	0T	B1 switch 2 (white)
PB2	15	B2 switch 3 (green)
PB3	0S	B3 switch 4 (red)
PB4	0Y	(B)
PB5	21	(B)
PB6	0Z	(B)
PB7	22	(B)

6522 Peripheral Interface Adaptor

ora4	A Conn	Function
PA0	21	D0 speech word
PA1	19	D1 speech word
PA2	0Y	D2 speech word
PA3	22	D3 speech word
PA4	20	D4 speech word
PA5	18	D5 speech word
PA6	0W	D6 speech word
PA7	17	D7 speech word

orb4	A Conn	Function
PB0	0K	18 address select line
PB1	0L	Tape audio in
PB2	0M	Tape audio out (LO)
PB3	0N	RCN-1 (1)
PB4	0P	Tape audio out (HI)
PB5	0T	TTY KB RTN (+)
PB6	N/C	
PB7	N/C	

APPENDIX B

SELECTOR AND DISTRIBUTOR PIN ASSIGNMENTS

DATA SELECTOR A

Input 00 - I/R proximity detector, left front

Input 01 - I/R proximity detector, right front

Input 02 - I/R proximity detector, left side

Input 03 - I/R proximity detector, right side

Input 04 - bumper impact, left front

Input 05 - bumper impact, right front

Input 06 - bumper impact, left side

Input 07 - bumper impact, right side

Input 08 - feeler impact, left side

Input 09 - feeler impact, right side

Input 10 - bumper impact, left rear

Input 11 - bumper impact, right rear

Input 12 - drive overload monitor

Input 13 - not used

Input 14 - I/R proximity detector, center front

Input 15 - I/R proximity detector, center rear

DATA SELECTOR B

Input 00 - ambient light

Input 01 - battery charge status

Input 02 - A/D overflow alarm

Input 03 - 9 volt bus monitor

Input 04 - 5 volt bus monitor

Input 05 - flooding alarm

Input 06 - smoke alarm

Input 07 - fire alarm

Input 08 - drive power switch monitor

Input 09 - selector check low

Input 10 - selector check high

Input 11 - vibration alarm

Input 12 - toxic gas alarm

Input 13 - approaching storm alarm

Input 14 - not used

Input 15 - not used

DATA SELECTOR C

Input 00	- random digit
Input 01	- hostile/friendly switch
Input 02	- alert verify
Input 03	- recharge monitor
Input 04	- I/R motion detector
Input 05	- optical target
Input 06	- optical range status
Input 07	- recharge probe status
Input 08	- center visual motion detector
Input 09	- right visual motion detector
Input 10	- left visual motion detector
Input 11	- bit 2 day of week
Input 12	- bit 1 day of week
Input 13	- bit 0 day of week
Input 14	- AM/PM status
Input 15	- parabolic I/R sensor

DATA DISTRIBUTOR A

Output 00 -	not used
Output 01 -	continuous spot enable
Output 02 -	transmitter enable
Output 03 -	alert and hold enable
Output 04 -	I/R interrupt enable
Output 05 -	strobe enable
Output 06 -	not used
Output 07 -	not used
Output 08 -	drive power relay
Output 09 -	not used
Output 10 -	low battery interrupt enable
Output 11 -	infrared motion detector enable
Output 12 -	visual motion detector enable
Output 13 -	position enable
Output 14 -	track enable
Output 15 -	scan enable

DATA DISTRIBUTOR B

Output 00 - not used

Output 01 - flood light timer trigger

Output 02 - not used

Output 03 - not used

Output 04 - not used

Output 05 - not used

Output 06 - not used

Output 07 - not used

Output 08 - not used

Output 09 - not used

Output 10 - not used

Output 11 - siren mode A trigger

Output 12 - siren mode B trigger

Output 13 - siren mode C trigger

Output 14 - not used

Ourput 15 - not used

APPENDIX C

INPUTS AVAILABLE TO CPU

AM/PM - 1 bit signal from a National Semiconductor Time/ Temperature Module (MA1026) indicates AM or PM. Also used to drive day of week counter.

Day of Week - a three bit binary code from counter, allowing CPU to establish what day it is (Sunday = 0, Monday = 1, etc.) for subsequent modification of behavior patterns.

Ambient light - signal from photocell on top of head which indicates if room is light or dark.

Temperature - sensing probe alerts CPU if ambient temperature exceeds or falls below adjustable set points.

Smoke - photoelectric smoke detection system.

Toxic gas - Figaro toxic gas detector.

Fire - infrared fire detector with backup mechanical heat sensor.

Vibration - seismic monitor indicates presence of vibration above an adjustable set point. Used for detection of earthquakes and/or physical contact from external source. This function gated out when robot is in motion.

Visual Motion Detection - three National Semiconductor D-1072 optical motion detection chips signal the CPU if ambient light levels increase or decrease more than 0.1 percent.

Infrared Motion Detection - Colorado Electro-Optics sensor mounted on the head detects changes in heat energy radiated by surroundings.

Optical Vision - a three photocell array mounted on the head and used for locating and tracking the beacon on top of the battery charging station. Optical Board Target Output goes high when any of the three photocells acquires the beacon.

Range - a special comparator compares the center photocell output with an adjustable set point. Used to indicate proximity of charger beacon.

Near-infrared Parabolic Sensor - mounted on the head and therefore positionable 100 degrees either side of center-line, this highly directional active sensor is used to establish the location of objects out to a distance of six feet. Provides excellent bearing resolution but gives no indication of range.

Flooding - spring loaded sensor indicates presence of water on floor.

Discriminatory Hearing - bandpass filter incorporated in sound activated circuitry to respond to high frequency noises such as breaking glass, sawing, or filing, for intrusion detection.

Head Position (relative) - analog to digital conversion, represents position of the head as a four-bit binary number.

Drive Status - six bit number which reflects the current steering command and drive direction status.

Battery Level - two separate comparators indicate when battery is in need of a charge, and when recharging is complete.

Sonar - LM1812 based forward looking sonar transceiver, used in collision avoidance routines when robot is in motion and for intrusion detection when platform is stationary.

Infrared Proximity Detectors - ten transmitter/receiver units employing active source high power near-infrared LEDs sense returned energy to indicate the presence of obstructions around the vehicle, out to a maximum range of 18 inches.

Contact Sensors - fourteen microswitches strategically positioned to indicate the deflection due to impact of spring loaded bumpers around the vehicle periphery.

Feelers - eight spring loaded feelers which sense object proximity for collision avoidance.

Bus Status Monitors - numerous comparators which monitor the voltage on various power distribution buses and initiate shutdown procedures in the event of a malfunction.

Storm Monitor - lightning discharges detected by an AM radio, output is rectified and fed to a capacitor. Voltage level across capacitor is monitored by a comparator which alerts CPU and activates a 24 hour weather broadcast receiver in the event of an approaching storm.

Probe Status - indicates presence of 14 volts on recharging probe when connected to battery charging station, activates relay to disable forward windings of tandem drive motors.

Drive Overload - Comparator monitors the voltage drop across the drive power circuit breaker for signs of stalled drive wheel.

Switch Position Verification - numerous comparators used to ensure critical switches are in correct position before initiating actions dependent on associated circuitry.

Speech Busy - 1 bit signal used to indicate to CPU completion of previously requested speech synthesis output.

Operator Input - four bit binary code manually loaded via toggle switches on Operator Control Panel to allow operator to request actions or modify behavior of robot.

ENTER Button - Normally open pushbutton to signal CPU that above switches have been set and are ready to read. Also used by operator to terminate current routine in execution.

Hostile/Friendly Switch - used by operator to advise robot as to action desired in event an intruder detected. In Friendly position the robot responds with a greeting of 'Hi' or 'Hello'. In hostile position the robot advises intruder to leave the room and then sets off alarm.

Alert Verify - used to confirm all previously powered down circuitry has come up on line as requested after transition from Passive Mode to Alert Mode.

APPENDIX D

PROXIMITY DETECTOR CIRCUITRY OPERATION

The near-infrared proximity detector system consists of a centrally located driver/detector board, with indicator LEDs, and remotely mounted transmitter/receiver units, relocatable for optimum placement. The driver circuitry is built up around two identical pulse generators, each producing a square wave train of 15 microsecond pulses with a pulse repetition period of 1.7 milliseconds. These pulses drive into saturation an NPN transistor which gates a XC-880-A high power gallium aluminum arsenide LED emitting energy in the near infrared spectrum (880 nanometers). The device is supplied in a T-1 3/4 package. A 47 mfd electrolytic and 10 ohm current limiting resistor are configured to supply an extremely heavy current flow (in excess of two amps) for the brief on-time, more than enough to destroy the LED under steady state conditions. The result is an intense pulsed output in a narrow cone, both desirable properties for an object detection system of this type. The two pulse generators are alternately enabled by a 555 astable multivibrator at about a 1 Hz rate, reducing power consumption by a factor of two, and eliminating pattern overlap of two adjacent LED's where desired. (There are certain cases where pattern overlap is used to shape and/or enhance the detection field of a sensor intentionally, as discussed later.)

The receivers utilized with this system each consist of a TIL413 photodiode incorporating a built in filter and lens system, with a cone shaped detection field roughly 45 degrees around the lens axis. The output of this photodiode is amplified through a L/C differentiator network, and then fed to a 741 (1/2 458) Op Amp (see Figure 28), which subsequently produces a positive spike for each burst of returned infrared energy detected. These pulses are inverted by a 4049, which also serves as a threshold detector, and used to trigger a 555 monostable (1/2 556). This acts as a pulse stretcher, providing an output pulse of approximately 100 ms, and illuminating a red LED for circuit monitoring and adjustment. The 555 monostable output generates an interrupt on IRQ Channel A and is then read by Data Selector A. Six receiver channels are provided, and all are commonly enabled or disabled by Data Distributor A output number 4 as needed. The circuitry is powered up automatically with the Drive Relay Board by Subroutine Dri.on. The receivers must subsequently be enabled by Subroutine I/Ren.

Position resolution of the detector is a function of receiver sensitivity, the photodiode field of view, and the irradiation pattern of the high power LED. Of these, the

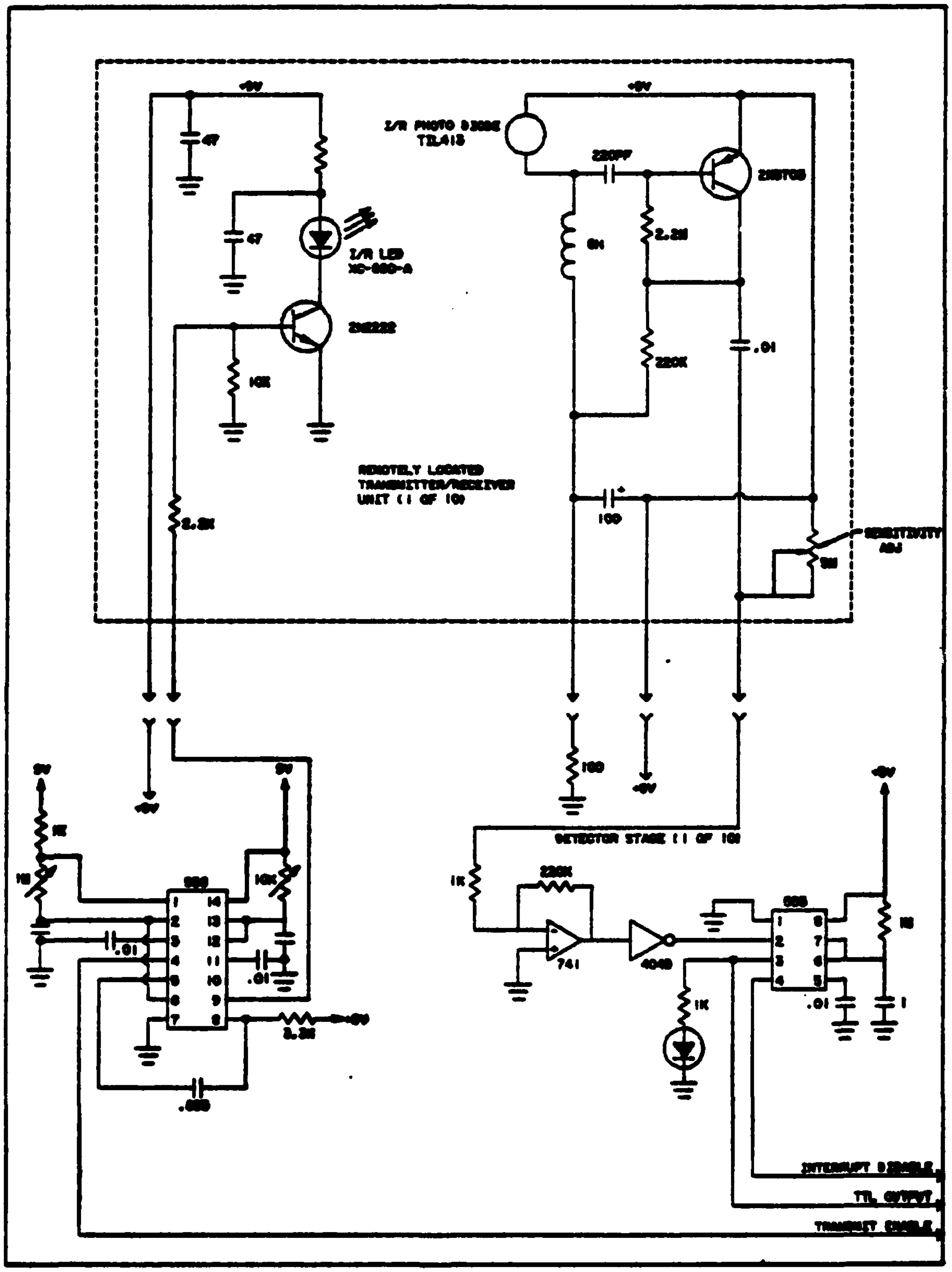


Figure 28. Near-infrared Proximity Detector Schematic

latter is the most significant determining factor, as the irradiation cone of the infrared LED is relatively narrow (40 degree angle between half power points) with respect to the detection cone of the photodiode. The usable irradiation cone angle was experimentally determined to be roughly half that of the half power angle, yielding good resolution of object location for a low cost system.

Where desired, multiple emitters can be used to advantage to strengthen the irradiation field for greater range or sensitivity, and through careful placement of the LEDs it is actually possible to shape the detection zone to best fit the application. The prototype robot is little concerned with how tall an object is, but rather interested in horizontal resolution of its exact location. Therefore emitters were arranged in vertical columns to expand the detection field vertically, while creating no horizontal overlap. This technique greatly expands the proximity detectors' versatility at minimal additional cost as long as the LED patterns remain within the photodiode field of view. A rough guideline to ensure reliability was found to be one field width either side of a normal LED irradiation pattern, as shown in Figure 29.

The near-infrared parabolic dish detector utilizes two adjacent LEDs for increased range and sensitivity, but the spatial resolution here is primarily dependent on the

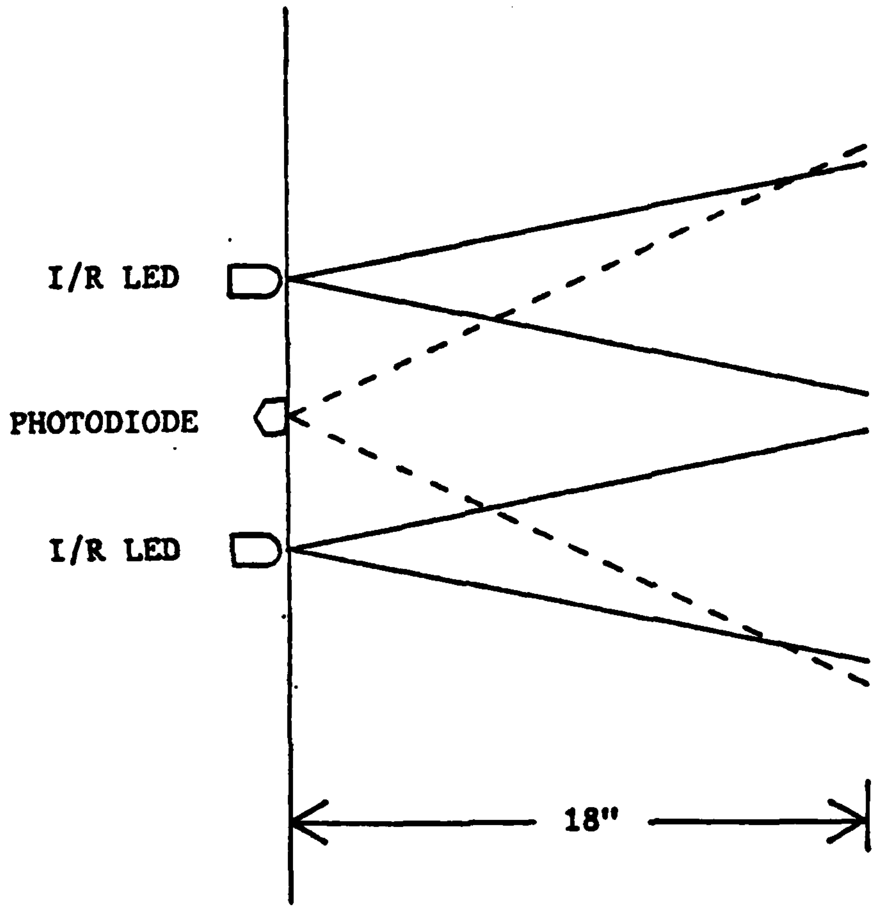


Figure 29. Near-infrared Sensor LED Overlap Pattern. Two emitters can be configured as shown to increase the vertical coverage of a single photodiode detector while maintaining the horizontal resolution required for collision avoidance.

directional property of the four inch parabolic reflector which focuses returned energy on the photodiode lens. The receiver circuitry is essentially the same as that employed in the proximity detectors, but with increased amplifier gain and an adjustable detector threshold implemented through use of an inverting comparator in place of the 4049 inverter gate. Additionally, the 555 monostable pulse stretcher is made retriggerable to provide a constant high output as long as pulses are received, free of intermediate resetting. This is a desirable property when the device is panned in search of a wall opening, for example, as a low output arises only when reflections stop, and not momentarily each time the 555 clocks out and resets.

APPENDIX E

POSSIBLE OPERATORS FOR ROBOT CONTROL LANGUAGE

Parameters follow operator and are separated by a space

SPEAK	Voice output word through speech synthesis
	First parameter is list number (1,2, or 3)
	Second parameter is word number on list
	Third parameter D causes 40 millisecond delay first
	First and second parameters are required
	Third parameter must be D or not used
	Ex: SPEAK 1 $4C output 'danger', list 1, word number 4C hex, no delay
	Ex: SPEAK 2 $0F D output 'circuit', list 2, word number 0F hex, delay 40 ms first
DELAY	Delay before continuing
	First parameter is number of half seconds of delay
	First parameter is required
	Second parameter C enables voice output of time of day if appropriate (see CLOCK) during wait
	Second parameter must be C or not used
	Ex: Delay 9 Wait 4.5 seconds before continuing Postpone announcement of time until authorized at later point
CLOCK	Allows announcement of time of day at this point in program if appropriate (hour or half hour)
	No parameters are used
POWER	Turns power on or off to interface and actuator circuitry
	First parameter must be ON or OFF
	Ex: POWER ON powers up system
SPOT	Turns spotlights on or off
	First parameter must be ON or OFF
SCAN	Controls automatic scan system which rotates head back and forth 95 degrees each side of center
	First parameter must be ON or OFF
TRACK	Enables tracking circuitry for locating beacon
	Also enables head to scan for lock on to beacon
	Disabled by SCAN OFF
	No parameters are used

```
INITIAL     Used to initialize system at start
            No parameters are used

DRIVE       Used to enable or disable drive control circuitry
            First parameter must be ON or OFF
            Ex: DRIVE ON    enables drive circuitry
                            does not start drive motor

POSITION    Used to position a joint
            First parameter must be joint identifier
            Second parameter must be desired position
            Ex: POSITION HEAD $08  sets head facing forward
            Ex: POSITION WHEEL $00     sets steering full right

HOME        Matches steering angle to head angle
            When used in a loop with tracking function enabled
              causes robot to home in on beacon
            No parameters are used

BEACON      Used to control beacon on top of recharging station
            First parameter must be ON or OFF

READ        Inputs data from specified source
            First parameter must be source identifier
            Second parameter can be used to further specify input
            Ex: READ HEAD       read head position (1 parameter)
            Ex: READ A 11       read Data Selector A, input no. 11

RANDOM      Creates a random integer 0 - 256
            First parameter is lower limit (inclusive)
            Second parameter must be upper limit (inclusive)
            No parameters returns random logic (high or low)
            Ex: RANDOM 20 200      create random integer in
                                   range 20 to 200
            Ex: RANDOM    randomly sets logical variable

CENTER      Determines head bearing to center of specified
            opening in wall, such as door or window
            First parameter specifies a bearing to left
              of desired opening to investigate
            Operation is then performed on first opening found
              to right of specified bearing
            Ex: CENTER $08  Find center of first opening located
                            to right of centerline ($08)

ALIGN       Causes robot to align itself with beacon dead ahead
              by backing, if such action requested by interrupt
              service routine, else no action taken
            No parameters are used
```

SKIRT	Causes robot to go around obstacle to left when homing on recharger, if such action requested. No parameters are used
MOVE	Controls motion of robot chassis First parameter must be drive motor command Legal choices are FORWARD, REVERSE, OPPOSITE, SAME, STOP Second parameter must be steering command Legal choices are LEFT, RIGHT, CENTER, SAME A variable (range 0 - 15) can also be used to specify intermediate steering angles Ex: MOVE FORWARD LEFT turn left moving forward Ex: MOVE OPPOSITE SAME reverse direction only Ex: MOVE REVERSE SAME move backward, steering same Ex: MOVE SAME RIGHT continue motion, turn right Ex: MOVE STOP LEFT turn wheel to left, drive off Ex: MOVE SAME $05 turn wheel to position $05
TEST	Perform a canned test routine as specified First parameter is system to be tested Ex: TEST HEAD perform canned test of head positioning circuitry Ex: TEST DRIVE perform canned test of drive wheel positioning circuitry
ALERT	Sets or secures system in 'alert' mode First parameter must be ON or OFF Ex: ALERT OFF secures system to 'passive' mode

LIST OF REFERENCES

1. SYM-1 Reference Manual, Synertek Systems Corporation, 1979.

2. Scanlon, L. J., 6502 Software Design, p. 180-218, 1980.

3. Zaks, R., 6502 Applications Book, p. 15-63, 1979.

4. Lancaster, D., TTL Cookbook, Sams, 1980.

5. DaCosta, F., How to Build Your Own Working Robot Pet, p. 86-87, Tab, 1979.

6. Frederiksen, T. M. and Howard, W. M., "A Single-Chip Monolithic Sonar System," IEEE Journal of Solid State Circuits, v. SC-9, No. 6, p. 394-402, December 1974.

7. Weiss, M., "Protect Your Valuables - Light Sensitive Security Alert," Radio-Electronics, April 1979.

8. IP-25 and IP-50 Installation Instructions, Colorado Electro-Optics, p. 3.

9. Weinstein, M. B., Android Design, p. 175-189. Hayden, 1981.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center	2
	Cameron Station	
	Alexandria, Virginia 22314	
2.	Library, Code 0142	2
	Naval Postgraduate School	
	Monterey, California 93940	
3.	Department Chairman, Code 69	1
	Department of Mechanical Engineering	
	Naval Postgraduate School	
	Monterey, California 93940	
4.	Keith L. Doty, Ph.D.	2
	Professor, Department of Electrical Engineering	
	225 Larsen Hall	
	University of Florida	
	Gainesville, Florida 32611	
5.	Urs Elsasser, Ph.D.	2
	Research Associate	
	Mechanical Engineering Department	
	Stanford University	
	Stanford, California 94305	
6.	Gary R. Cook	1
	Program Manager	
	Physical Security Research and Development	
	USN Civil Engineering Laboratory	
	Construction Battalion Center	
	Port Hueneme, California 93043	
7.	Raymond P. Lutz, Ph.D.	1
	Executive Dean of Graduate Studies and Research	
	University of Texas at Dallas	
	P.O. Box 688	
	Richardson, Texas 75080	
8.	Russell Richards, Ph.D., Code 55Rh	1
	Professor, Department of Operations Research	
	Naval Postgraduate School	
	Monterey, California 93940	

9.	Gordon E. Latta, Ph.D., Code 53Lz Professor, Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
10.	Robert E. Newton, Ph.D., Code 69Ne Professor, Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93940	2
11.	LCDR Hobart R. Everett, USN Special Assistant for Robotics Naval Sea Systems Command Navy Department Washington, DC 20362	10
12.	Department of Electrical Engineering Georgia Institute of Technology Atlanta, Georgia 30332	1
13.	Officer in Charge of Robotics and Artificial Intelligence Naval Ocean Systems Center San Diego, California 92152	1